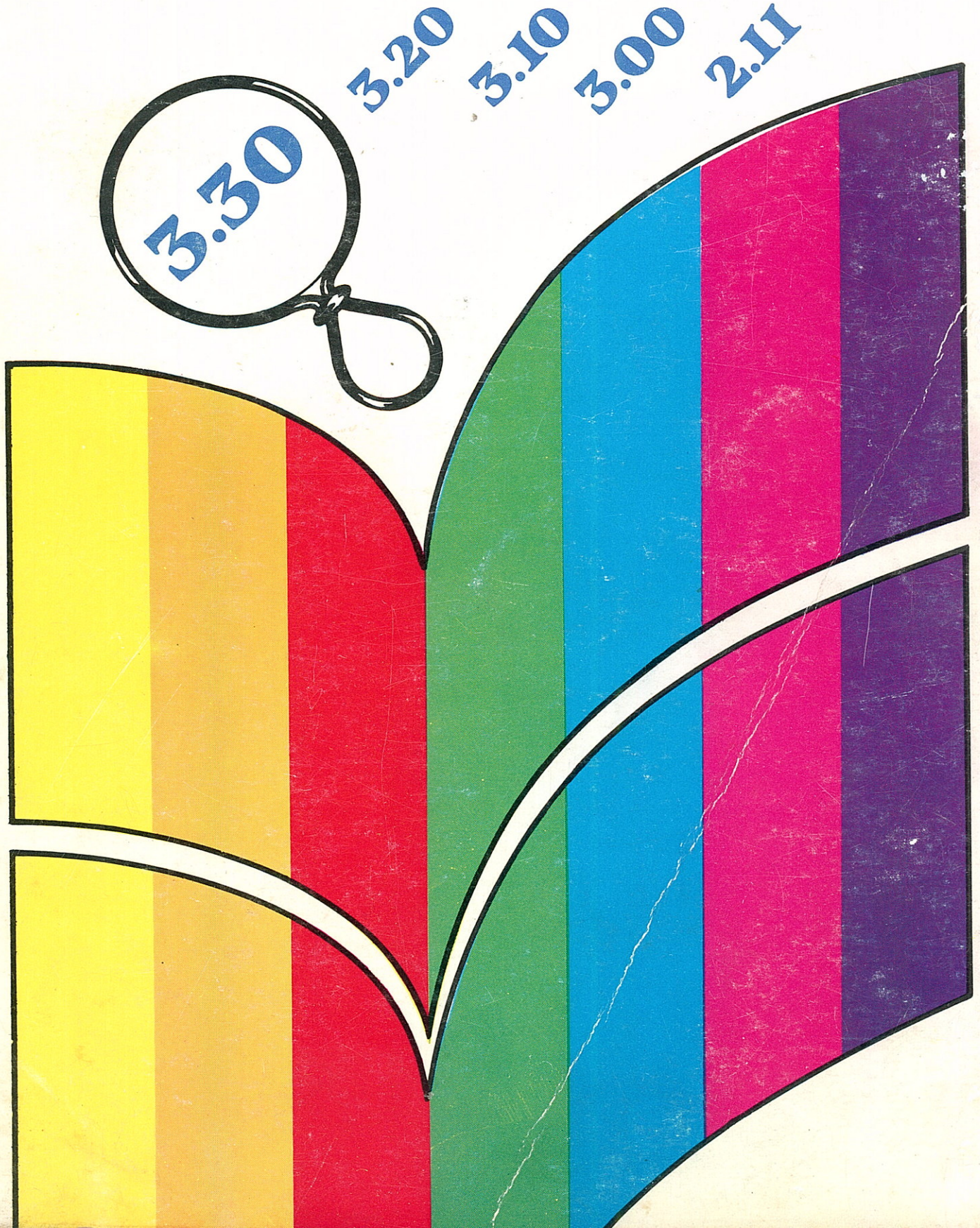
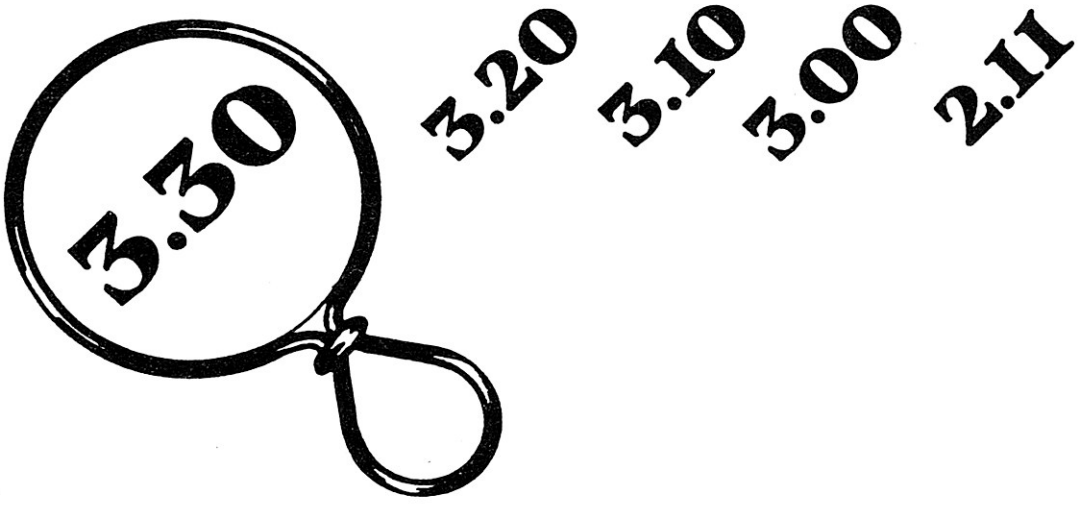
IBM PC DOS

III. PC DOS PROGRAMOZÓI SEGÉDLET



IBM PC DOS

III. PC DOS PROGRAMOZÓI SEGÉDLET



LSI
ALKALMAZÁSTECHNIKAI TANÁCSADÓ SZOLGÁLAT
BUDAPEST, 1988

Szerkesztette: Dr. Úry László III.

Készítette: Orosz Judit
Baki Zoltán
Pajor Gábor

Lektorálta: Éltető László

Kiadó: LSI Alkalmazástechnikai Tanácsadó Szolgálat
Felelős kiadó: dr. Kovács Magda
Témafelelős: Székely László
Szedés: LSI ATSz Info Osztály
Technikai szerkesztő: Nagy Olivér
Engedélyszám: 57152
ISBN összkötet: 963 592 814 9
ISBN I. kötet: 963 592 811 4
ISBN II. kötet: 963 592 812 2
ISBN III. kötet: 963 592 813 0
Készült a Szabadság MgTsz Nyomdaüzemében, Gyál 88-282/3
Felelős vezető: Tóth Antal

Előszó

Ez a kézikönyv az IBM PC XT/AT-kompatibilis mikroszámítógépek egyik leghatékonyabb programfejlesztő eszközének, a macro assembler-nek használatához szükséges ismereteket foglalja össze.

A DOS-kézikönyv csak a megszakítások és funkcióleírások megértéséhez szükséges legfontosabb információkat foglalta össze a 8086 mikroprocesszor-családról, jelen könyv 2. fejezete részletesen ismerteti a processzor felépítését és az utasítások struktúráját.

A 3.-5. fejezetek a forrásnyelvű assembly programok felépítésével foglalkoznak, részletesen tárgyalják az assembly utasításokat és használható direktívákat.

A további fejezetek magáról a MASM macro assembler fordítóprogram, illetve a kapcsolódó segédprogramok használatát mutatják be. Ezek közé tartozik a LIB programkönyvtár-kezelő program, a LINK szerkesztő, a CREF kereszthivatkozás-listát készítő program, valamint a MAKE és a programok nyomkövetését biztosító szimbólum hibakereső program, a SYMDEB.

A függelék az assembler és a szerkesztő hibaüzeneteit tartalmazza.

A Kiadó

Tartalomjegyzék

1. BEVEZETÉS 10

2. A MIKROPROCESSZOR-CSALÁD ARCHITEKTÚRÁJA 11
2.1 A 8088 CPU blokkdiagramja 11
2.2 Regiszterek 12
2.3 Tárcímzés szegmentálással 12
2.4 Adatcímzési módok 13
2.5 Az utasításkészlet fő csoportjai 15
2.5.1 Adatátviteli utasítások 15
2.5.2 Aritmetikai utasítások 16
2.5.3 Logikai utasítások 16
2.5.4 Karaktersorozat-kezelő utasítások 18
2.5.5 Vezérlésátadó utasítások 19
2.5.6 Processzorvezérlő utasítások 22

3. ADATTÍPUSOK 23
3.1 Karakterkészlet 23
3.2 Numerikus konstansok 23
3.2.1 Egész számok 23
3.2.2 Valós számok 24
3.2.3 Kódolt valós számok 24
3.2.4 Tömörített decimális számok 24
3.3 Karakter- és karaktersorozat-konstansok 25
3.4 Azonosítók 25
3.5 Operandusok 26
3.5.1 Konstans operandusok 26
3.5.2 Direkt címmel elérhető operandusok 26
3.5.3 Relatív címmel elérhető operandusok 27
3.5.4 Számláló operandus 27
3.5.5 Regiszter-operandusok 27
3.5.6 Bázis-relatív operandusok 28
3.5.7 Indexelt operandusok 29
3.5.8 Bázis-relatív indexelt operandusok 29
3.5.9 Struktúra-operandusok 30
3.5.10 Rekord-operandusok 30
3.5.11 Rekordmező-operandusok 30
3.6 Az operandusokból előállítható kifejezések 31
3.6.1 Aritmetikai műveletek 31
3.6.2 Léptető műveletek 32
3.6.3 Logikai műveletek 32
3.6.4 Indexelés 32
3.6.5 A PTR művelet 34
3.6.6 A címkiszámítás viszonyítási alapjának megváltoztatása 34

3.6.7 A SHORT művelet 35
3.6.8 A THIS művelet 35
3.6.9 A HIGH és a LOW művelet 36
3.6.10 A SEG művelet 36
3.6.11 Az OFFSET művelet 36
3.6.12 A TYPE művelet 37
3.6.13 A .TYPE művelet 37
3.6.14 A LENGTH művelet 38
3.6.15 A SIZE művelet 39
3.6.16 A WIDTH művelet 39
3.6.17 A MASK művelet 39
3.6.18 A kifejezés értékének kiszámítása és a precedencia 40
3.6.19 A memória-operandusok típusának ellenőrzése 40

4. A MIKROPROCESSZOR-CSALÁD ASSEMBLY UTASÍTÁSAI 41
4.1 Az utasítások leírásában használt rövidítések 41
4.2 A 8086 és a 8088 utasításkészlete 42
4.3 A 8087 utasításkészlete 87
4.3.1 A 8087-es numerikus adatprocesszor felépítése 87
4.4 A 80186 és a 80286 utasításkészlete 93
4.5 A 80287 utasításkészlete 95

5. PROGRAMOZÁS MAKRO-ASSEMBLY NYELVEN 96
5.1 A makró-assembly programok felépítése 96
5.1.1 A makró-assembly utasítások általános alakja 96
5.1.2 Az utasításkészletet beállító direktívák 96
5.1.3 A forrás-modulok és az END direktíva 97
5.1.4 A SEGMENT és az ENDS direktíva 98
5.1.5 Az ORG direktíva 99
5.1.6 Az EVEN direktíva 99
5.1.7 A PROC és az ENDP direktíva 100
5.1.8 A GROUP direktíva 100
5.1.9 Az ASSUME direktíva 101
5.1.10 A megjegyzések és a COMMENT direktíva 101
5.1.11 Példák a makró-assembly programok felépítését meghatározó direktívák használatára 102
5.2 Deklarációk 103
5.2.1 Adatdeklarációk 103
5.2.2 Címkedeklarációk 108
5.2.3 Szimbólumdeklarációk 108
5.2.4 Struktúrák deklarálása 110
5.2.5 Rekordok deklarálása 113
5.2.6 Előre-hivatkozások 115
5.2.7 Globális deklarációk 116
5.3 Makrók definiálása és hívása 119
5.3.1 A MACRO és az ENDM direktíva 119

5.3.2 Makróhívások 121
5.3.3 Részkaraktersorozat megadása makró-paraméterrel 121
5.3.4 A szöveg-konstans jele 123
5.3.5 A karakter-konstans jele 123
5.3.6 A kifejezés jele 123
5.3.7 A PURGE direktíva - makródefiníciók törlése 124
5.3.8 A LOCAL direktíva - egyedi azonosítók létrehozása makrókban 124
5.3.9 A REPT és az ENDM direktíva - utasításcsoport ismétlése 125
5.3.10 Az IRP és az ENDM direktíva - utasításcsoport ismétlése paraméterhelyettesítéssel 126
5.3.11 Az IRPC és az ENDM direktíva 127
5.3.12 Az EXITM direktíva 128
5.3.13 Makró-megjegyzések 128
5.3.14 Feltételes fordítás 129
3.14.1 Az IF és az IFE direktíva 130
3.14.2 Az IF1 és az IF2 direktíva 130
3.14.3 Az IFDEF és az IFNDEF direktíva 130
3.14.4 Az IFB és az IFNB direktíva 131
3.14.5 Az IFIDN és az IFDIF direktíva 131
5.4 A makró-assembler file-kezelésének vezérlése 132
5.4.1 Az INCLUDE direktíva 132
5.4.2 A TITLE direktíva 133
5.4.3 A SUBTITLE direktíva 133
5.4.4 A PAGE direktíva 133
5.4.5 A .LIST és a .XLIST direktíva 134
5.4.6 A .LALL, a .XALL és a .SALL direktíva 134
5.4.7 A .SFCOND, a .LFCOND és a .TFCOND direktíva 135
5.4.8 A .RADIX direktíva 135
5.4.9 A .CREF és a .XCREF direktíva 136
5.4.10 A .NAME direktíva 136
5.4.11 A %OUT direktíva 137
5.5 Magasszintű nyelvből hívott Assembly rutinok 137

6. A MASM FORDÍTÓ HASZNÁLATA 138
6.1 Bevezetés 138
6.2 Forrás file fordítása 138
6.3 A MASM kapcsolói 139
6.3.1 /D, az első menet listája 139
6.3.2 /ML, kis-nagybetűk megkülönböztetése a lokális szimbólumokban 139
6.3.3 /MX, kis-nagybetűk megkülönböztetése a globális szimbólumokban 139
6.3.4 /X, a feltételes fordítások hamis ágának listázása 140
6.3.5 /R, kódképzés az aritmetikai processzor számára 140
6.3.6 /E, kódképzés a lebegőpontos emulátor számára 140
6.3.7 /A, szegmensek abc sorrendben 140
6.4 A lista-file 140
6.4.1 A forrássorok 141

6.4.2 A makrótáblázat 141
6.4.3 A struktúra-, rekordtáblázat 141
6.4.4 A szegmens-, csoporttáblázat 142
6.4.5 A szimbólumtáblázat 142
6.4.6 Az első menet listája 143

7. PROGRAMKÖNYVTÁR-KEZELÉS A LIB SEGÉDPROGRAMMAL 144
7.1 A programkönyvtár és kezelése 144
7.2 A LIB funkciói 144
7.2.1 A LIB hívása parancs-sorral 146
7.2.2 A LIB használata párbeszédes módban 146
7.2.3 LIB parancs-sorozat használata file-ból (válasz-file) 148
7.3 Új könyvtár létrehozása 149
7.4 A lapméret átállítása (PAGESIZE) 149
7.5 A könyvtár konzisztenciájának ellenőrzése 150
7.6 A keresztreferencia-lista és készítése 150

8. FUTTATHATÓ PROGRAM ELŐÁLLÍTÁSA - Programszerkesztés (LINK) 151
8.1 A LINK indítása 151
8.1.1 Programszerkesztés PC DOS parancs-sorral 152
8.1.2 Programszerkesztés interaktív üzemmódban 154
8.1.3 Programszerkesztés előre összeállított válasz-file-val 154
8.1.4 Könyvtárak elérési útjainak megadása 155
8.1.5 A map file szerkezete 155
8.1.6 Ideiglenes lemez-file - VM.TMP 156
8.2 A LINK opciói 157
8.2.1 Megszakítás a szerkesztés alatt - PAUSE 157
8.2.2 Public szimbólumok táblájának előállítása - MAP 158
8.2.3 A verem-méret meghatározása - STACK 158
8.2.4 A program helyigényének maximuma - CPARMAXALLOC 186
8.2.5 A kezdőcím maximumra állítása - HIGH 160
8.2.6 Helyfoglalás az adatcsoportnak - DSALLOCATE 160
8.2.7 Forrás-sorok kezdőcímének kiírása - LINENUMBERS 160
8.2.8 Kis- és nagybetűk megkülönböztetése - NOIGNORECASE 160
8.2.9 A könyvtár-alapfeltételezés törlése - NODEFAULTLIBRARYSEARCH 161
8.2.10 Csoportok eltávolítása a programból - NOGROUPASSOCIATION 116
8.2.11 Megszakítás overlay esetén - OVERLAYINTERRUPT 161
8.2.12 A szegmensek maximális számának meghatározása - SEGMENTS 162
8.2.13 Az MS-DOS szegmens-elrendezés alkalmazása - DOSSEG 162
8.3 A LINK működése 162
8.3.1 A szegmensek címhatárra való igazítása 163
8.3.2 A szegmensek kezdőcímének meghatározása 163
8.3.3 A szegmensek átrendezése 163
8.3.4 Kombinált szegmensek 164
8.3.5 Szegmens-csoportok 164
8.3.6 Címfeloldás 164

8.3.7 A betöltés sorrendjének vezérlése 165

9. PROGRAMOK KARBANTARTÁSA A MAKE PROGRAM SEGÍTSÉGÉVEL 167
9.1 Bevezetés 167
9.2 A MAKE használata 167
9.3 MAKE leíró file-ok létrehozása 167
9.4 A MAKE futtatása 168
9.5 MAKE opciók 168
9.6 Makró definíciók 169
9.7 Példa a MAKE használatára 169

10. SZIMBÓLIKUS NYOMKERESŐ ÉS HIBAKÖVETŐ PROGRAM (SYMDEB) 170
10.1 Bevezetés 170
10.2 A SYMDEB indítása 170
10.2.1 Indítás program file-lal 171
10.2.2 Indítás szimbólumokkal 171
10.2.3 Paraméterátadás a betöltendő programnak 171
10.2.4 Indítás file nélkül 171
10.2.5 Szimbólum file készítése (MAPSYM) 171
10.3 A kontroll karakter funkciója 172
10.3.1 Parancs megszakítása 172
10.3.2 Parancs felfüggesztése 172
10.3.3 Az NMI használata 172
10.4 A parancsok használata 172
10.4.1 A parancsok általános felépítése 172
10.4.2 Szimbólumok 173
10.4.3 Számok 173
10.4.4 Címek 173
10.4.5 Cím-tartomány 173
10.4.6 Sorszámok 173
10.4.7 Karaktersorozatok 174
10.4.8 Kifejezések 174
10.5 A SYMDEB parancsai 175
10.5.1 Az Assemble (fordítás) parancs 175
10.5.2 A BreakPoint Set (töréspont megadása) parancs 175
10.5.3 A Breakpoint Clear (töréspont törlése) parancs 176
10.5.4 A Breakpoint Disable (töréspont kikapcsolása) parancs 176
10.5.5 A Breakpoint Enable (töréspont bekapcsolása) parancs 176
10.5.6 A Breakpoint List (töréspont kiírása) parancs 176
10.5.7 A Compare (összehasonlítás) parancs 176
10.5.8 A Display ? (kifejezés értékének kiírása) parancs 177
10.5.9 A Dump ASCII parancs 177
10.5.10 A Dump Bytes parancs 177
10.5.11 A Dump Words parancs 177
10.5.12 A Dump Doublewords parancs 177
10.5.13 A Dump Short Reals parancs 177

10.5.14 A Dump Long Reals parancs 177
10.5.15 A Dump Ten-Byte Reals parancs 178
10.5.16 A Dump parancs 178
10.5.17 Az Enter parancs 178
10.5.18 Az eXamine Symbol Map parancs 178
10.5.19 A Fill parancs 178
10.5.20 A Go parancs 178
10.5.21 A Help ? parancs 179
10.5.22 A Hex parancs 180
10.5.23 Az Input parancs 180
10.5.24 A Load parancs 180
10.5.25 A Move parancs 181
10.5.26 A Name parancs 181
10.5.27 Az Open Map XO parancs 181
10.5.28 Az Output parancs 181
10.5.29 A PTrace parancs 181
10.5.30 A Quit parancs 181
10.5.31 A Redirection <,>,= parancs 181
10.5.32 A Registers parancs 182
10.5.33 A Search parancs 182
10.5.34 A Set Source Mode parancs 182
10.5.35 A Trace parancs 183
10.5.36 Az Unassemble parancs 183
10.5.37 A Write parancs 183
10.6 Hibaüzenetek 183
10.7 A SYMDEB kompatibilis fordítók 184

11. **KERESZTHIVATKOZÁSI-LISTA KÉSZÍTŐ PROGRAM (CREF)** 185
11.1 Bvezetés 185
11.2 A CREF HASZNÁLATA 185
11.3 A kereszthivatkozási-lista 186
11.4 Hibaüzenetek 186

12. **HIBAJELZÉSEK** 187
12.1 Az assembler hibaüzenetei 187
assembler hibák 187
I/O hibák 192
12.2 A szerkesztő hibaüzenetei 192

1. Bevezetés

A macro-assembler a 8086/80186/80286 mikroprocesszor-család utasításainak fordítására készült. A 8086, 8088, 8087, 80186, 80286 és 80287 (későbbi változatai a 80386) mikroprocesszorok utasításait tudja feldolgozni.

A 8086 és a 8088 mikroprocesszorok funkcionális szempontból lényegében azonosak. A köztük lévő szerkezeti eltérés a működési sebességben nyilvánul meg: a 8086 gyorsabb.

A 8087 és a 80287 az ún. lebegőpontos koprocesszorok.

A 80186 és a 80286 mikroprocesszorok a 8086 utasításkészletét bővítik további utasításokkal.

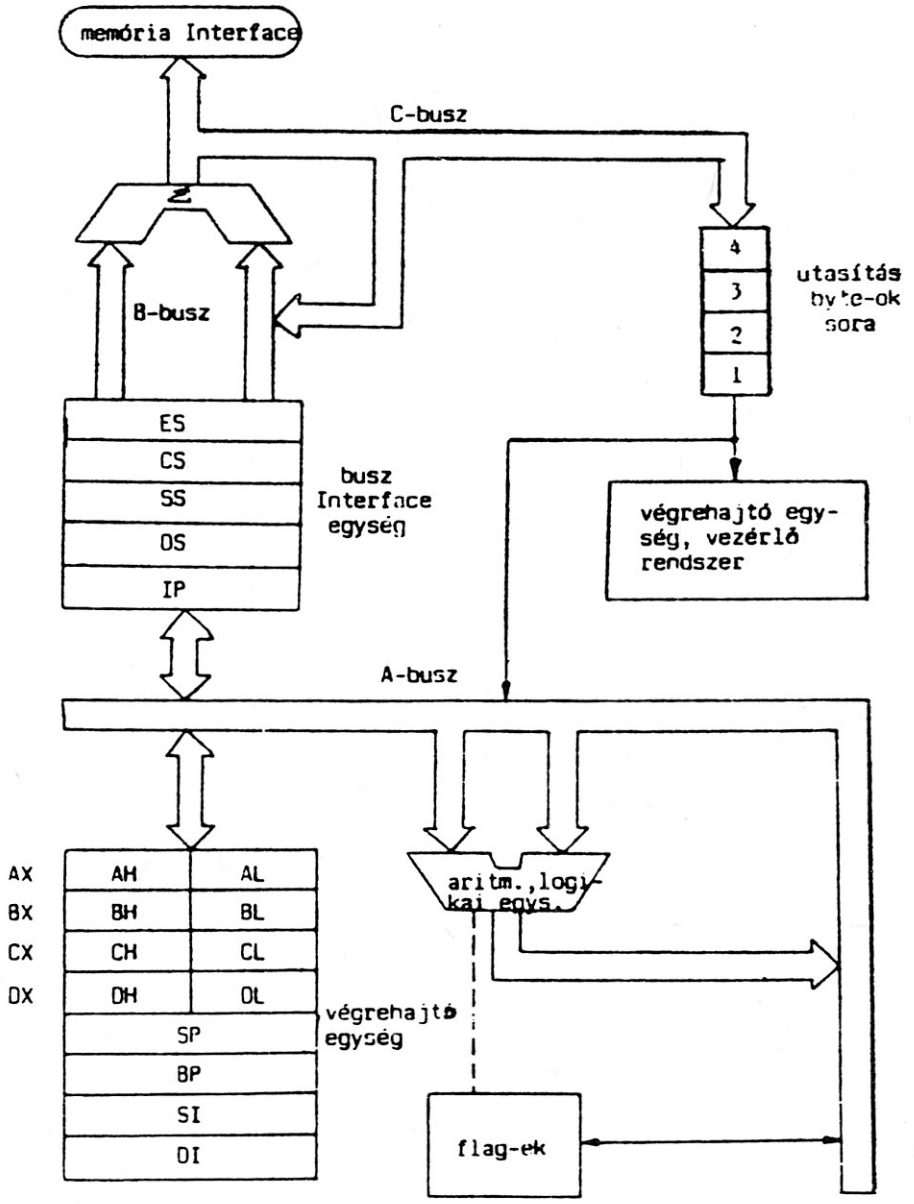
A macro-assembler alapértelmezésben csak a 8086/88 és a 8087 utasításait ismeri fel. Ha a forrásprogram a 80186, 80286 vagy 80287 utasításokat tartalmaz, akkor a programban meg kell adni a megfelelő utasításkészletet beállító direktívá(ka)t.

A könyvben használt jelölések

[ ]	A szögletes zárójelek között lévő elemet nem kötelező megadni.
...	Azt jelzi, hogy az előtte lévő elem ismételhető.
,,,	Azt jelzi, hogy az előtte lévő elem ismételhető, és az egyes ismételt elemeket vesszővel kell elválasztani egymástól.
VASTAG NAGYBETŰ	Utasítás vagy paraméter neve, amelyet pontosan a megadott formában kell használni.
dőltbetű	Olyan azonosító, amelyet a programozó maga határozhat meg.
|	Az egymástól függőleges vonallal elválasztott elemek közül pontosan egyet kell megadni. Ha ezeket a lehetőségeket szögletes zárójelpárba tettük, akkor az el is maradhat. Például [+ | –] azt jelenti, hogy ezen a helyen semmi, a + vagy a – jel szerepel.

2. A mikroprocesszor-család architektúrája

2.1 A 8088 CPU blokkdiagramja



A processzor és a memória közötti adatforgalom a memória interface-en keresztül bonyolódik le. A processzorhoz érkező adatok az utasítás byte-ok sorába kerülnek. A végrehajtó egység vezérlő rendszere kiveszi az utasításokat ebből a sorból és a végrehajtó egységnek továbbítja, amely értelmezi és végrehajtja azokat. Ameddig ez a folyamat zajlik, a busz interface egység megpróbálja újra feltölteni az utasítás byte-ok sorát. Ez lényegesen megrövidíti az utasítások végrehajtási idejét, mivel - a legtöbb mikroprocesszortól eltérően - a következő végrehajtandó utasítás már a processzorba került a memóriából, nem kell várni a beolvasásra.

A végrehajtó egység aritmetikai/logikai egységet is tartalmaz, amely a számításokat és az összehasonlításokat végzi. A flag-ek (jelző bitek) az eredmény állapotát jelzik (pl. nulla, negatív).

2.2 Regiszterek

Programozói szempontból a regiszterek a mikroprocesszor legfontosabb alkotóelemei, mivel az assembly nyelvű programban állandóan hivatkozni kell rájuk. Mindegyik regiszter 16 bit hosszúságú.

Négy **általános célú regiszter** van: AX, BX, CX és DX. Ezek felső vagy alsó byte-jára külön is lehet hivatkozni. A felső (magasabb helyiértékű) byte-ot az AH, BH, CH, DH az alsó (alacsonyabb helyiértékű) byte-ot az AL, BL, CL, DL regiszternév jelöli.

Az SI és DI regiszterek a karaktersorozatok indexregiszterei. SI-ben a **küldő mező indexét**, DI-ben a **fogadó mező indexét** szokás tárolni. SI és DI általános regiszterként is használható.

Az SP veremmutató regiszter lehetővé teszi hardware verem implementálását a memóriában. Az SP mindig a verem tetején lévő elemre mutat. A BP bázis-regiszterrel a veremben lévő adatokat érhetjük el. A BP regisztert általános regiszterként is lehet használni.

Négy szegmensregiszter használatos a memória címzéséhez: CS, DS, SS és ES. A mikroprocesszor mindig hivatkozik egy vagy több szegmensregiszterre, amikor egy utasítást végrehajt.

Az IP regiszter utasításmutató regiszter, amely a következő végrehajtandó utasítás címét tartalmazza.

A flag-ek regisztere egybites információkat tartalmaz a mikroprocesszor aktuális állapotáról. Ezek leírása a II. fejezet 5. pontjában található.

2.3 Tárcímzés szegmentálással

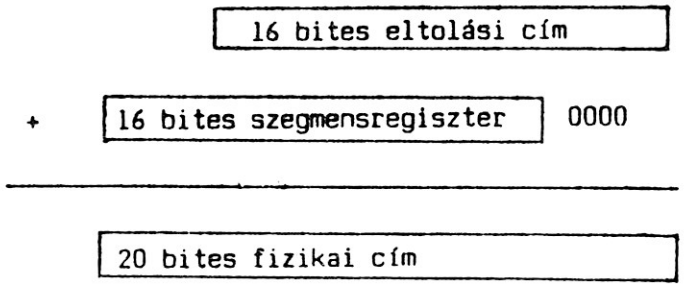
A 8086 mikroprocesszor család 16-bites címekkel hivatkozik a központi memóriára. Egy 16-bites címmel legfeljebb 64 Kbyte-ot (=65536 byte-ot) lehet megcímezni. A memória szegmensekre bontásával és a szegmensek címzésével a tárkapacitás 1 Mbyte-ig (1048576 byte) bővíthető.

1 Mbyte címzéséhez 20-bites címszóra van szükség. A memóriaszegmensek legfeljebb 64K méretűek lehetnek, ezért a szegmensen belüli címzés 16 biten megoldható.

Minden szegmensnek paragrafushatáron kell kezdődnie, ami azt jelenti, hogy valamennyi szegmens kezdőcíme 16-tal osztható. A szegmensek átlapolhatók, vagyis egy adott byte több szegmens kezdőcímén keresztül is elérhető lehet.

A mikroprocesszor úgy számítja ki a memória tényleges címét, hogy először kiválasztja azt a szegmensregisztert, amelyben az objektumot tartalmazó szegmens címe található. Ezt az értéket 4 bit-tel balra lépteti (megszorozza 16-tal), így megkapja a szegmens fizikai kezdőcímét, majd ehhez hozzáadja az objektum szegmens-relatív, más néven eltolási

címét. Az eredmény az objektum teljes, 20-bites fizikai címe. A következő ábra ezt a műveletet mutatja.



A szegmensregiszterek a következő szegmenstípusok kezdőcímét tartalmazzák:

CS: CODE szegmens, amelyben a végrehajtandó utasítások találhatók

DS: DATA szegmens, amelyben adatok illetve sztringkezelő utasítások input adatmezői vannak

SS: STACK szegmens, amelyet a veremkezelő műveletek végrehajtásakor használunk; az elemekhez való hozzáférést a BP bázisregiszter biztosítja

ES: EXTRA szegmens, amely másodlagos adatterületet és sztringkezelő műveletek output adatmezőit foglalhatja magába.

Egy utasítás címének kiszámításakor a processzor a CS regiszterben lévő szegmenscímet és az IP utasításszámlálóban lévő eltolási címet használja fel.

Az eddigiekből következik, hogy egyidejűleg legfeljebb 4x64 Kbyte = 256 Kbyte tárterületet lehet elérni.

Ha a program kevés adattal dolgozik, nincs szükség EXTRA szegmensre. Ilyenkor ES tartalma a DS tartalmával egyenlő, tehát az EXTRA szegmens azonos az adatszegmenssel.

Minthogy a szegmensregiszterek írhatók és olvashatók, a szegmensek elhelyezését dinamikusan lehet változtatni.

2.4 Adatcímzési módok

Az adatokra következőképpen hivatkozhatunk:

- regiszterrel: pl. BX vagy CH
- konstanssal vagy literállal: pl. 5 vagy 'C'
- közvetlen memóriacímmel: pl. ADAT
- indirekt címzéssel: pl. [BX]
- bázis-relatív címzéssel: pl. [BX + 5]
- bázis-relatív indexelt címzéssel: pl. [BX + SI + 5]

Regiszterrel való hivatkozáskor az adat az adott 8 vagy 16 bites regiszterben van.

A **konstanssal** vagy **literállal** való hivatkozás azt jelenti, hogy az adatot beírjuk az utasításba.

Közvetlen memóriacímmel egy azonosító által kijelölt memóriaterület tartalmára hivatkozunk.

Indirekt címzés esetén az adott terület tartalma címként értelmeződik, és az e címen lévő memóriatartalomra hivatkozunk.

Ha **bázis-relatív címzést** adunk meg, akkor a bázisregiszter tartalmának és az eltolásnak az összege adja azt a tárcímet, amelyen a hivatkozott adat van.

Bázis-relatív indexelt címzés esetén az adat fizikai címe a bázisregiszter tartalmának, az indexregiszter tartalmának és az eltolásnak az összege.

Az indirekt, bázis-relatív és bázis-relatív indexelt címzésekben a BX, BP, SI és DI regiszterekre lehet hivatkozni.

Konstans vagy literál csak küldő mezőként szerepelhet az utasításokban. A többi címzési módot akár küldő, akár fogadó mezők esetében lehet használni az utasítás típusától függően.

Ha egy tárbeli operandusra hivatkozunk, akkor a processzor azt feltételezi, hogy az adat a DATA szegmensben van és a címkiszámításkor a DS szegmensregiszter tartalmát veszi figyelembe. *Egyetlen kivétel ez alól az az eset, amikor a BP regisztert bázisregiszterként használjuk*: ekkor az SS regiszteren keresztül a verem-szegmensre hivatkozunk. Ezért ha pl. az elérendő adat az EXTRA szegmensben van, akkor ezt külön jelölni kell: pl. ES:[BX].

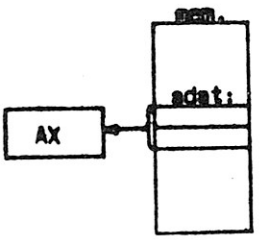
Példák:

A következő példákban olyan műveleteket ábrázolunk, amelyek az AX regiszterbe egy egyszavas adatot töltenek be.

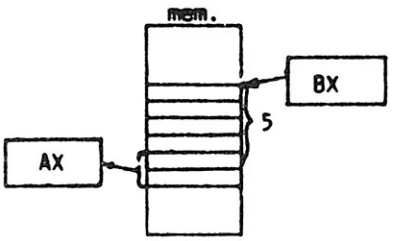
művelet / címzési mód / működés

MOV AX,BX
regiszter

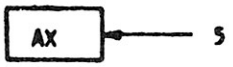
MOV AX,ADAT
közvetlen memória



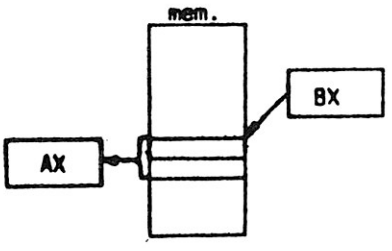
MOV AX,[BX + 5]
bázis-relatív



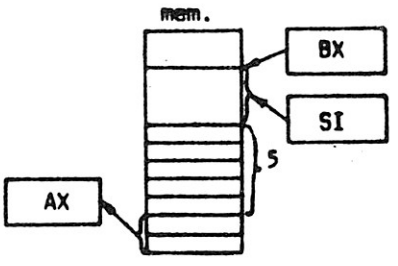
MOV AX,5
konstans



MOV AX,[BX]
indirekt



MOV AX,[BX + SI + 5]
bázis-relatív indexelt



2.5 Az utasításkészlet fő csoportjai

Az **utasításkészlet** azon elemi műveletek összessége, amelyeket a mikroprocesszor végre tud hajtani. Ezeket funkció szerint 6 fő csoportba soroljuk:

- adatátviteli utasítások
- aritmetikai utasítások
- logikai utasítások
- karaktersorozat-kezelő utasítások
- vezérlésátadó utasítások
- processzorvezérlő utasítások.

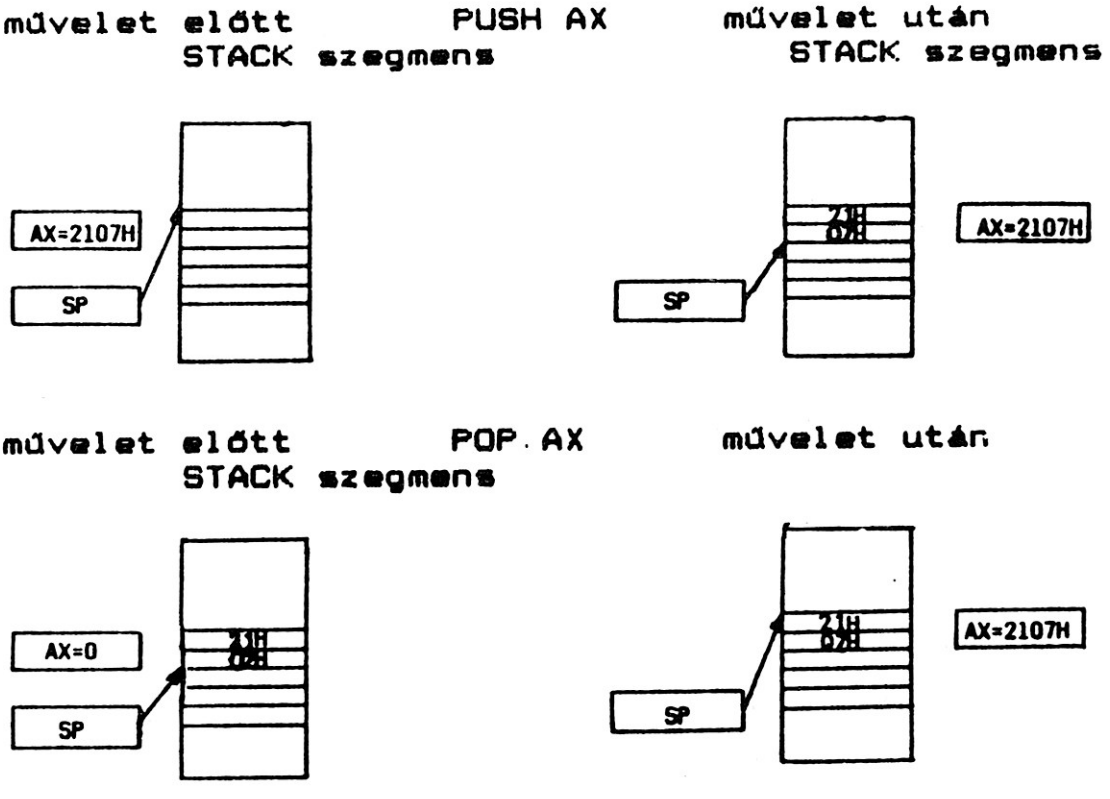
Ebben a fejezetben az utasítások működésének lényegét írjuk le. Pontos szintaktikájuk a III. fejezetben található.

2.5.1 Adatátviteli utasítások

Az adatátviteli utasítások a következők:

- operandus feltöltése: MOV
- veremkezelő műveletek: POP, PUSH
- két operandus tartalmának felcserélése: XCHG
- input/output műveletek: IN, OUT
- konvertálás: XLAT

A **veremkezelő műveletek** közül a PUSH adatot helyez el a veremben, a POP pedig adatot vesz ki a veremből. *Mindig szavas adatműveletet végeznek.* Működésüket a következő ábrák szemléltetik:



A veremből az utoljára betett adatot lehet először kivenni. A **veremkezelő műveleteket** általában a program adatainak mentésére, szubrutinhíváskor és a megszakítási technika kezelésekor szokás használni.

Az **input/output műveletek** i/o portokon (kapukon) keresztül biztosítják a mikroprocesszor és a külső készülékek (terminál, nyomtató, lemez) közötti kommunikációt. Az i/o portok byte-os vagy szavas adatok átvitelére alkalmasak. Az adatokat az AX regiszterből kell elküldeni (OUT) ill. az AX regiszterbe kerülnek (IN). Ha az adat csak egy byte-os, akkor elegendő az AL regiszter használata. Az i/o port címét a DX regiszter tartalmazza, vagy az utasításban konstansként/literálként adjuk meg. Az utóbbi esetben csak az első 256 i/o port érhető el.

A **konvertálást végző XLAT utasítás** úgy működik, hogy a BX regiszter tartalmához hozzáadja az AL regiszter tartalmát, majd az eredményt címként értelmezve az e címen lévő byte-ot betölti az AL regiszterbe. Általában úgy használják, hogy BX tartalma egy konverter-tábla kezdőcíme.

2.5.2 Aritmetikai utasítások

Az aritmetikai utasítások a következők:

- összeadás: ADD
- inkrementálás: INC
- kivonás: SUB
- dekrementálás: DEC
- kettes komplemens képzése: NEG
- összehasonlítás: CMP
- szorzás: MUL
- osztás: DIV

Ezeket a műveleteket 1 byte vagy 1 szó hosszúságú operandusokon lehet végezni. Minden aritmetikai utasítás az elvégzett művelet eredményétől függően beállítja a flag-regiszter megfelelő bitjeit. A feltételes vezérlésátadó utasításokkal a flag-ek tartalma lekérdezhető és programelágaztatást lehet végrehajtani.

2.5.3 Logikai utasítások

A logikai utasítások a következők:

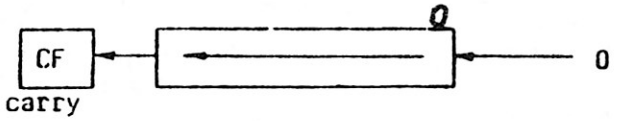
- tagadás: NOT
- "és" művelet: AND
- "vagy" művelet: OR
- "kizáró vagy" művelet: XOR
- byte vagy szó bitjeinek tesztelése: TEST
- léptetés balra: SHL, SAL
- léptetés jobbra: SHR, SAR
- ciklikus léptetés balra: ROL, RCL
- ciklikus léptetés jobbra: ROR, RCR

A logikai utasítások nem numerikus értékeknek, hanem bitsorozatoknak tekintik az operandusokat.

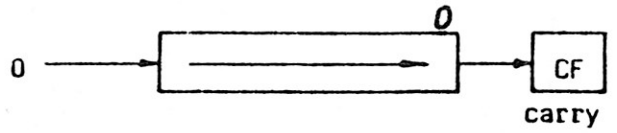
A NOT, AND, OR, XOR és TEST logikai utasítások mindig törlik a carry és az overflow flag-et és a zero flag-et 1-re állítják, ha a művelet eredménye 0. (A TEST is logikai "és" műveletet hajt végre, akárcsak az AND, de az operandusok változatlanok maradnak.)

Ha a bitsorozat numerikus értéket ábrázol, akkor a balra és jobbra léptető utasításokat 2 hatványaival való műveletvégzésre is használhatjuk. A balra léptetéssel szorzást, a jobbra léptetéssel osztást végezhetünk. Működésüket a következő ábra mutatja:

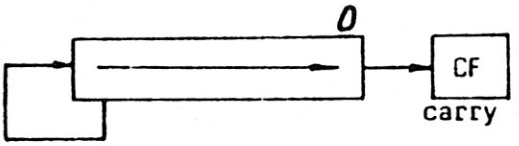
balra léptetés (SHL)



jobbra léptetés (SHR)



Ha a léptetendő operandus előjeles numerikus érték (2-es komplemens formában ábrázolva), akkor a jobbra léptetést, azaz az osztást aritmetikai léptetésként is végezhetjük (SAR utasítás). Ekkor a bal szélső (legmagasabb helyiértékű) bit változatlan marad. Az aritmetikai jobbra léptetés működési ábrája:



A ciklikus léptető utasítások esetében a kilépő bitek nem "vesznek el", hanem 0 helyett ezekkel töltődnek fel a szabaddá váló bitek.

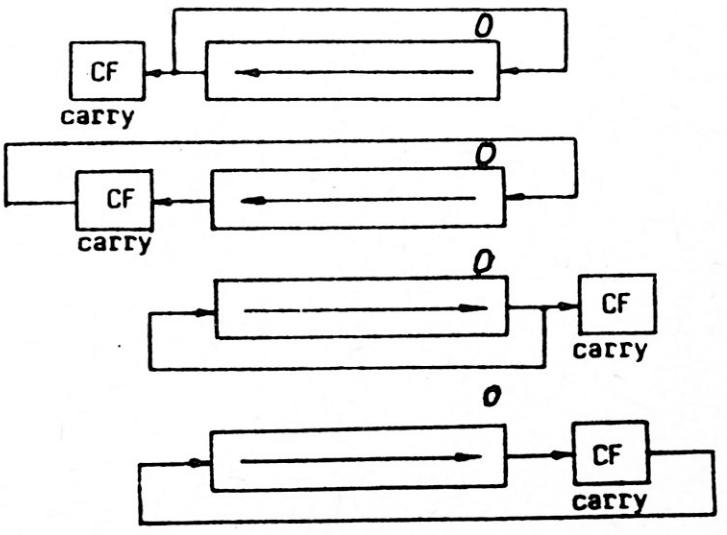
Működésük:

ciklikus balra léptetés (ROL)

ciklikus balra léptetés a carry flag-en keresztül (RCL)

ciklikus jobbra léptetés (ROR)

ciklikus jobbra léptetés a carry flag-en keresztül (RCR)



A logikai műveletek operandusai 1 byte vagy 1 szó hosszúságúak lehetnek.

2.5.4 Karaktersorozat-kezelő utasítások

A karaktersorozatok kezelését könnyítő utasítások a következők:

- karaktersorozat másolása: MOVS
- két karaktersorozat összehasonlítása: CMPS
- adott byte vagy szó keresése egy karaktersorozatban: SCAS
- karaktersorozat egy byte-jának vagy egy szavának betöltése az AL vagy az AX regiszterbe: LODS
- az AL vagy az AX regiszter tartalmából karaktersorozat előállítása: STOS.

Mindegyik karaktersorozat-kezelő utasítás 1 byte vagy 1 szó hosszúságú operandusokkal működik.

A **karaktersorozat-kezelő utasítások** operandusait nem az utasításban adjuk meg, hanem megállapodás szerinti helyeken. Eszerint a küldő mező elemeit az SI regiszterrel címezzük, a fogadó mező elemeit a DI regiszterrel. A **fogadó mezőnek** mindig az EXTRA szegmensben kell lennie. A **küldő mező** alapértelmezés szerint a DATA szegmensben helyezkedik el. Ha másik szegmensben lévő küldő mezőre akarunk hivatkozni, akkor explicit módon meg kell adni a mezőt tartalmazó szegmens szegmensregiszterét. A küldő és a fogadó mező azonos szegmensben is lehet. Ilyenkor a DS és az ES szegmensregiszterek tartalmának azonosnak kell lennie.

A karaktersorozat-kezelő műveleteket a kisebb memóriacímtől a nagyobb felé haladva, vagy fordított irányban, a nagyobb címtől a kisebb felé haladva lehet végrehajtani. **A feldolgozás iránya az irányt jelző flag értékétől függ**: ha a flag értéke 0, akkor a címek növekvő sorrendjében történik a feldolgozás, ha a flag értéke 1, akkor a címek csökkenő sorrendjében.

A karaktersorozat-kezelő művelet végrehajtása előtt a DS és az SI regiszter a küldő mező kezdőcímét, az ES és a DI regiszter a fogadó mező kezdőcímét határozza meg. A karaktersorozat-kezelő utasítás automatikusan inkrementálja vagy dekrementálja SI és DI értékét a feldolgozás irányától függően.

A karaktersorozat egy byte-jának vagy szavának az AL illetve az AX regiszterbe való töltése kivételével a string-kezelő utasítások ismételhetők (REP, REPE, REPNE). Ilyenkor a karaktersorozat hoszszát a CX regiszterben kell elhelyezni. A művelet végrehajtása során CX tartalma automatikusan csökken.

Az AL vagy AX tartalmának tárolásakor az SI tartalma változatlan marad, csak DI módosul, mert nincs forrás-karaktersorozat. A művelet végrehajtása előtt CX az előállítandó karaktersorozat hosszát tartalmazza, a művelet végrehajtása után CX tartalma nulla lesz.

Egy karaktersorozat egy byte-jának vagy egy szavának betöltésekor az AL illetve az AX regiszterbe a DI tartalma változatlan marad, csak SI módosul, mert nincs fogadó karaktersorozat.

Ha egy adott byte-os vagy szavas értéket akarunk keresni egy karaktersorozatban, akkor a keresett értéket AL-ben vagy AX-ben helyezzük el. Az SCAS utasítás kivonja AL vagy AX tartalmából az ES és DI regiszterrel címzett byte-ot vagy szót. A műveletvégzés során

csak az összehasonlítás eredményét jelző flag, és a DI regiszter tartalma módosul. Az összehasonlítást vagy akkor hagyjuk abba, amikor egyezést találunk (REPNE), vagy akkor, amikor eltérést találunk (REPE). Az ismétlési tényezőt a keresés előtt CX tartalmazza. A keresés végrehajtása után CX tartalma azoknak a byte-oknak vagy szavaknak a száma, amelyek a string kezdetétől haladva a keresett byte-ot vagy szót megelőzték.

A két karaktersorozatot összehasonlító utasítás rendre kivonja a forrás-mező byte-jaiból vagy szavaiból a fogadó mező byte-jait vagy szavait. A művelet hatására az operandusok nem változnak, csak az összehasonlítás eredményét jelző flag, valamint SI és DI tartalma módosul. Az összehasonlítást vagy akkor hagyjuk abba, amikor egyezést találunk (REPNE), vagy akkor, amikor eltérést találunk (REPE). Az összehasonlítás ismétlési tényezőjét a műveletvégzés előtt CX tartalmazza. Műveletvégzés után CX tartalma azoknak a byte-oknak vagy szavaknak a száma, amelyek nem kerültek összehasonlításra.

Az SI illetve DI regisztert módosító karaktersorozat-kezelő műveletek végrehajtása után SI illetve DI az utoljára feldolgozott byte vagy szó után soron következő címre mutat, figyelembe véve a feldolgozás haladási irányát.

2.5.5 Vezérlésátadó utasítások

Mint már említettük, a proceszor byte-onként olvassa be a memóriából a végrehajtandó utasításokat. A következő végrehajtandó utasítás mindig a CS regiszter tartalma által meghatározott aktuális kódszegmensnek az utasítás-számláló regiszter (IP) által meghatározott relatív címén van. A program végrehajtása általában az utasítások felirási sorrendjében történik. Minden utasítás végrehajtása után az IP regiszter tartalma megfelelő mértékben növekszik..

A vezérlésátadó utasítások lehetővé teszik, eltérjünk a szekvenciális programfutástól. Ennek két alapvető módja van:

- az IP regiszter tartalmának megváltoztatásával ugyanabban a kódszegmensben, de más relatív címen folytatódik a program futása - ezt **szegmensen belüli vezérlésátadásnak** nevezzük.
- a CS és az IP regiszter tartalmának egyidejű megváltoztatásával egy másik kódszegmensben levő címen folytatódik a program futása - ez a **szegmensek közötti vezérlésátadás**.

A következő vezérlésátadó utasításokat lehet használni:

- feltétel nélküli ugrás: JMP
- feltételes ugrások: JZ, JNZ,...
- feltétel nélküli ciklus: LOOP
- feltételes ciklus: LOOPE, LOOPNE,...
- szubrutinhívás: CALL
- szubrutinból való visszatérés: RET
- megszakítás: INT
- megszakításból való visszatérés: IRET

Feltétel nélküli ugrást szegmensen belül vagy szegmensek között lehet végrehajtani. Szegmensen belüli vezérlésátadás esetében az utasítás operandusaként megadott cím lehet:

- direkt
  - 16 bittel címezhető, a szegmensen belül tetszőleges helyen levő címke;
  - 8 bittel címezhető, a vezérlésátadó utasítástól 128 byte-nál kisebb távolságra lévő címke (SHORT típusú ugrás)
- indirekt
  - egy regiszter, amely címet tartalmaz
  - egy memóriacím, amely címet tartalmaz.

Szegmensek közötti vezérlésátadás esetében az utasítás operandusaként megadott cím lehet:

- direkt: egy másik szegmensben deklarált címkeé
- indirekt: egy memóriacím, amelyen lévő szó tartalma az IP regiszterbe, az ezt követő szó tartalma a CS regiszterbe kerül.

A feltételes ugró utasítások hatására akkor következik be vezérlésátadás, ha az utasításkóddal meghatározott feltétel teljesül. A feltételek a jelzőbitek (flag-ek) tartalmára vonatkoznak. A flag-ek álttalában az aritmetikai és logikai utasítások eredményét jelzik. A zero, sign, overflow és parity flag közvetlenül tesztelhető. A feltételes vezérlésátadó utasításokat csak szegmensen belüli, direkt címzési módú és 128 byte-nál kevesebb távolságú ugrásokra lehet használni.

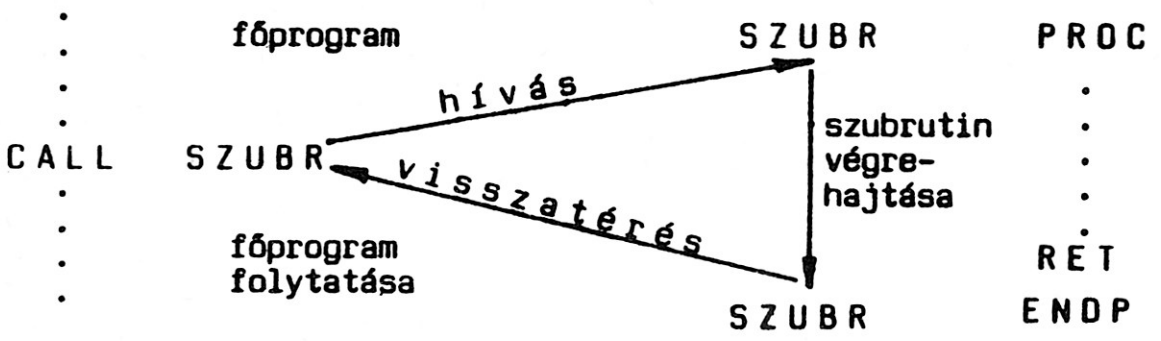
Az előbb említett megkötés (szegmensen belüli, direkt címzési mód, 128 byte-nál kisebb távolság) a **ciklusszervező utasításokra** is érvényes. A ciklusszervező utasítások lehetővé teszik, hogy egy utasítássorozat végrehajtását megismételjük.

Az ismétlendő sorozat előtt a CX regiszterben kell tárolni az ismétlési tényezőt, majd címkével kell ellátni a ciklusmag első utasítását. A ciklusmag minden végrehajtása után a CX tartalma automatikusan csökken. Háromféle ciklust lehet szervezni:

- akkor fejeződik be az ismétlés, ha CX = 0 (LOOP);
- akkor fejeződik be az ismétlés, ha CX = 0 vagy a zero flag értéke 0 (LOOPE, LOOPZ);
- akkor fejeződik be az ismétlés, ha CX = 0 vagy a zero flag értéke 1 (LOOPNE, LOOPNZ).

A ciklus befejezése után a program végrehajtása a ciklusszervező utasítást követő utasítással folytatódik.

A szubrutinok olyan utasítássorozatok, amelyek a program tetszőleges pontjánál hívhatók. A CALL utasítás hatására a vezérlés a szubrutin első utasítására kerül. A szubrutin utolsó utasítása a RET, amely visszaadja a vezérlést a hívó program CALL utasítását követő első utasítására. Ezt a folyamatot a következő ábrán láthatjuk:



A főprogram és a szubrutin közötti paraméterátadást a vermen keresztül szokás lebonyolítani.

Szegmensen belüli és más szegmensben definiált szubrutin egyaránt hívható. A CALL utasítás operandusára ugyanazok a szabályok vonatkoznak, mint a feltétel nélküli ugró utasítás (JMP) operandusára.

A CALL végrahajtásakor a visszatérési cím automatikusan a verem tetejére kerül. A RET végrehajtásakor a processzor kiveszi a visszatérési címet a veremből. *A verem használata lehetővé teszi a szubrutinok egymásba ágyazását és biztosítja az utolsó hívás helyére való visszatérést.*

Az I/O eszközökről a mikroprocesszorhoz érkező jeleket **megszakításoknak** (interrupt) nevezzük. A megszakításokat a mikroprocesszor **interrupt-mechanizmusa szolgálja ki**, amely két alapvető lehetőséget nyújt:

- letilthatja, hogy a mikroprocesszor érzékelje a megszakításokat, ezt az interrupt flag 0-ra való állításával lehet elérni, kivéve a nem maszkolható megszakításokat (NMI), amelyek érzékelése és kezelése az interrupt flag tartalmától független;
- a megszakítás fogadása után menti a mikroprocesszor aktuális állapotát, így a megszakítás kiszolgálása után folytatható az abbahagyott tevékenység (az aktuális állapot mentése úgy történik, hogy a flag-regiszter tartalma a verembe kerül, az interrupt és a trap flag törlődik, majd a CS és az IP regiszter tartalma is a verembe kerül.

A processzor aktuális állapotának mentése után a megszakítási mechanizmus a beérkezett megszakításoknak megfelelő kiszolgáló rutinra adja a vezérlést. Ha ez a rutin lefutott (azaz a megszakítást kezelő rutin egy IRET utasítást hajt végre), akkor az IRET hatására a processzor kiveszi a veremből az IP-t, a CS-t, és a flag-ek regiszterét, és a vezérlés arra a programra kerül, amelyet a processzor a megszakítás pillanatában hajtott végre.

A megfelelő kiszolgáló rutin kiválasztása az INT utasítás operandusa, egy 1 byte-os típuskód alapján történik. Ez azt jelenti, hogy legfeljebb 256 kiszolgáló rutin hívása lehetséges. A programban az 5-255 típuskódok adhatók meg, mivel a 0-4 típuskódot a mikroprocesszor saját magának foglalja le:

0 - a **nullával való osztás** kezelése
1 - **nyomkövetés**
2 - **nem maszkolható megszakítás**
3 - **speciális 1 byte-os megszakítások** (nincs típuskód operandusa az INT utasításnak)
4 - **túlcsordulás** kezelése (INTO utasítás)

2.5.6 Processzorvezérlő utasítások

A **processzorvezérlő utasítások** az átviteli, a feldolgozás irányát jelző és a megszakítás flag-et állítják be. A HLT utasítás felfüggeszti a processzor működését. Ezt akkor használjuk, ha egy megszakításra való várakozás ideje alatt semmilyen tevékenységet nem akarunk végrehajtani.

A flag-eket állító processzorvezérlő utasítások a következők:

- a carry flag törlése: CLC
- a direction flag törlése: CLD
- az interrupt flag törlése: CLT
- a carry flag ellekezőjére állítása: CMC
- a carry flag 1-re állítása: STC
- a direction flag 1-re állítása: STD
- az interrupt flag 1-re állítása: STI.

3. Adattípusok

3.1 Karakterkészlet

Minden assembly nyelvű program **utasításokból** és **megjegyzésekből** áll. Az utasítások kialakításához a következő **karakterkészlet** elemeit lehet használni:

ABCDEFGHIJKLMNOPQRSTUVWXYZ (angol nagybetűk)

abcdefghijklmnopqrstuvwxyz (angol kisbetűk)

0123456789 (számjegyek)

?@ $:.[]() < > {}

+ -/*&%!_ ‡\ = # ;,'" (speciális jelek és az aláhúzás)

3.2 Numerikus konstansok

3.2.1 Egész számok

Alakjuk:

számjegy ...[típusjel]

Az egész szám bináris, oktális, decimális vagy hexadecimális számjegy(ek)ből és opcionális típusjelből áll. Ha a típusjelet nem adjuk meg, akkor a fordító az aktuális alapfeltételezést használja, ami a program indításakor a decimális típus. *A típus alapértelmezését a .RADIX direktívával változtathatjuk meg.* A típusjelet a programban akár nagy-, akár kisbetűvel meg lehet adni. A .RADIX direktíva részletes leírása a IV. fejezet 4.8 pontjában található.

Az egész számban megadható jegyek maximális száma attól függ, hogy milyen utasításban vagy direktívában használjuk.

A következő táblázat azt mutatja be, hogy melyik típus milyen számjegyeket tartalmazhat:

Típusjel	Típus	Számjegyek
B	bináris	0 1
Q	oktális	0 1 2 3 4 5 6 7
O	oktális	0 1 2 3 4 5 6 7
D	decimális	0 1 2 3 4 5 6 7 8 9
H	hexadecimális	0 1 2 3 4 5 6 7 8 9 A B C D E F

A hexadecimális számoknak mindig decimális számjeggyel (0-9) kell kezdődniük. A hexadecimális jegyek (A-F) nagy- és kisbetűvel is írhatók.

Példák:

101B 173Q 4FH 69D 69 0FH 9ABH

3.2.2 Valós számok

Alakjuk:

számjegy... . számjegy... [**E**[**+** | **-**]*számjegy...*]

A valós számok egész részből, törtrészből és esetleg kitevőből állnak. A számjegyek decimális jegyek. A kitevő használata nem kötelező.

Valós számokat csak a DD, DQ és DT direktívában lehet használni. A szám jegyeinek maximális száma és a kitevő maximális értéke az adott direktívától függ.

A DD, DQ és DT adatdeklaráló direktívák pontos leírása a IV. fejezet 2.1 pontjában található.

Példák:

98.76 9.876E1 9876.E-2

3.2.3 Kódolt valós számok

Alakjuk:

számjegy ... **R**

A **kódolt valós szám** egy 8-, 16- vagy 20-jegyű hexadecimális szám. A kódolt valós szám előjelmezőből, kitevőből és mantisszából áll. Ezek az értékek bitmezőkként tárolódnak a memóriában. A bitmezők pontos mérete és jelentése a szám bitjeinek számától függ. A számnak decimális jeggyel kell kezdődnie (0-9).

Kódolt valós számokat csak a DD, DQ és DT direktívában lehet használni. A jegyek száma vezető nullával való megadás esetén pontosan 9, 17 vagy 21 kell hogy legyen.

Példák:

3F000000R
3F00000000000000R

3.2.4 Tömörített decimális számok

Alakjuk:

[**+** | **-**]*számjegy...*

A **tömörített decimális számok** decimális egészek, amelyek tömörített decimális formában tárolódnak. Tömörített decimális tárolás esetén az előjel-byte-ot 9 érték-byte követi. Minden érték-byte 2 decimális jegyet tartalmaz. Az előjel-byte legmagasabb helyiértékű bitje 0, ha a szám pozitív, és 1, ha a szám negatív. Mindig 10 byte-on ábrázolódnak.

A tömörített decimális számok olyan alakúak, mint a decimális egészek, attól eltekintve, hogy előjelesek is lehetnek és csak a DT direktívában használhatóak. A tömörített decimális számok legfeljebb 18-jegyűek lehetnek.

Példák:

```
97540      ; kódolva: 0000000000000097540
-97540     ; kódolva: 8000000000000097540
```

3.3 Karakter- és karaktersorozat-konstansok

Alakjuk:

'karakter...'
"karakter..."

A karakterkonstans egyetlen ASCII karakterből áll. A karaktersorozat-konstans két vagy több ASCII karakter kombinációja. Mindkét konstanst aposztrofok (') vagy idézőjelek (") közé kell tenni.

Ha a határoló jel a konstans belsejében is szerepel, akkor ott kettőzni kell.

Példák:

```
'x' "x"
'cdef'
"paraméterhiba"
'ilyen "kulcs" nem létezik' de "ilyen 'kulcs' nem létezik"
```

3.4 Azonosítók

Alakjuk:

karakter...

Az **azonosító betűk, számjegyek és speciális karakterek kombinációja**, amelyek címkeként, változóként vagy szimbólumként lehet használni.

Az azonosítóra a következő formai szabályok vonatkoznak:

1. Betűvel vagy a _ ? $ @ speciális karakterek valamelyikével kell kezdődnie.
2. Az azonosítóban vegyesen szerepelhetnek kis- és nagybetűk. A kisbetűk nagybetűkké konvertálódnak, ha nem használjuk a /ML vagy /MX opciót a fordító elindításakor.
3. Az azonosító tetszőleges hosszúságú lehet, de a macro-assembler csak az első 31 karaktert veszi figyelembe.

Példák:

```
HIBA2
_alprogram
Feldolgozás
```

3.5. Operandusok

A macro-assembly program utasításaiban és direktíváiban különböző operandusokat használhatunk. Az operandus konstans, címke, változó vagy egyéb szimbólum lehet, amely az utasítás vagy direktíva által végzendő műveletben értéket, regisztert vagy memóriaterületet jelöl.

Az utasításokban és a direktívákban használható operandus-típusok:

- konstans
- direkt címmel elérhető adat
- relatív címmel elérhető adat
- számláló
- regiszter
- bázis-relatív
- indexelt
- bázis-relatív indexelt
- struktúra
- rekord
- rekordmező

3.5.1 Konstans operandusok

Alakjuk:

szám | *karaktersorozat* | *kifejezés*

A konstans a többi operandustól eltérően nem memóriacímet jelent, hanem konkrét értéket, amelyet egy számmal, egy karaktersorozat-konstanssal, egy szimbólummal vagy egy kifejezéssel adhatunk meg. A kifejezések felépítésére vonatkozó tudnivalók a II. fejezet 6. pontjában találhatók.

Példák:

szám: 123
karaktersorozat-konstans: 'abcd'
szimbólum: alma
kifejezés: 5*7

3.5.2 Direkt címmel elérhető operandusok

Alakjuk:

szegmens:eltolás

A direkt címmel elérhető operandus a tár egy vagy több bytejának abszolút címét adja meg. A "szegmens" szegmensregiszternév (**CS, DS, SS, ES**), szegmensnév vagy csoport neve lehet. Az eltolás egy olyan érték, amely 0 és 65535 közé esik és amelyet egész számmal, abszolút szimbólummal vagy kifejezéssel lehet megadni.

Példák:

DS:26
adatok:26
csoport:elem

3.5.3 Relatív címmel elérhető operandusok

Alakjuk:

szimbólum

A relatív címmel elérhető operandus olyan szimbólum, amely egy utasítás vagy adat memóriacímét jelenti. A direkt címmel elérhető operandustól eltérően annak a szegmensnek vagy csoportnak a kezdetéhez relatív, amelyben az adott szimbólumot definiáltuk. Konkrét értéket csak a programszerkesztés folyamán kap.

Példa:

cím

3.5.4 Számláló operandus

A számlálóra a $ jellel hivatkozhatunk. A számláló a fordítás során az aktuális szegmens aktuális helyét jelenti. Ez egy olyan utasításcím, amely az aktuális szegmens kezdetéhez relatív. Az eltolást a fordító számolja ki oly módon, hogy a szegmens minden utasításnak lefordítása után annyival növeli a számláló értékét, ahány byte-ot generált. Az operandus végleges értékét csak a szerkesztés folyamán kapja meg.

3.5.5 Regiszter-operandusok

Alakjuk:

regiszternév

A regiszternév egy processzor-regiszter neve, amely a következők valamelyike lehet:

AX	**AH**	**AL**	**BX**	**BH**	**BL**
CX	**CH**	**CL**	**DX**	**DH**	**DL**
CS	**DS**	**SS**	**ES**	**SP**	**BP**
DI	**SI**				

A regiszterekre való hivatkozáskor nagy- és kisbetűk tetszőleges kombinációja megengedett.

Az AX, BX, CX és DX 16-bites általános célú regiszterek, amelyeket aritmetikai vagy logikai műveletek operandusaiként használhatunk. Az általános regiszterek alsó és felső 8 bitjére külön-külön is lehet hivatkozni. A megfelelő regiszterek alsó 8 bitjére való hivatkozást AL, BL, CL és DL, a felső 8 bitjére való hivatkozást AH, BH, CH és DH jelöli.

CS, DS, SS és ES szegmensregisztert azonosít. CS a kódszegmens, DS az adatszegmens, SS a veremszegmens, ES a különleges szegmens címét tartalmazza. Minden utasítás és adat címe a szegmensregiszterek valamelyikében lévő címhez relatív.

Az SP regiszter 16-bites és a verem-mutatót (stack pointer) tartalmazza, amely a verem tetejének aktuális címe. Ez a cím az SS regiszterben lévő szegmenscímhez relatív. A vermet használó utasítások automatikusan módosítják az SP regiszter tartalmát.

BX, BP, DI és SI 16-bites bázis- és indexregiszterek. Ezeket az általános regisztereket a program adataihoz való hozzáféréskor használhatjuk.

A 16-bites flag-regiszternek nincs azonosítója. A regiszter tartalma a LAHF, SAHF, PUSHF és POPF utasítások használatával hozzáférhető. A flag-regiszter kilenc egybites flag-et tartalmaz, amelyek helye és jelentése a következő:

Pozíció Jelentés

0 Carry (átvitel): aritmetikai műveleteknél a legmagasabb helyiértékű biten keletkezett átvitel.

2 Parity (paritás): értéke 1, ha egy 8-bites adattal végzett művelet eredményeképpen páros számú 1-es bit keletkezett, egyébként 0.

4 Auxiliary carry (segéd-átvitel): 8-bites adattal végzett művelet esetén a 3. és 4. bit közt keletkezett átvitel.

5 Trap (nyomkövetés): ha értéke 1, a processzor nyomkövető üzemmódban működik (azaz minden utasítás után egy INT 1 megszakítást generál).

6 Zero (nulla eredmény): értéke 1, ha egy aritmetikai, vagy logikai művelet eredménye 0.

7 Sign (előjel): aritmetikai művelet végrehajtásakor keletkezett eredmény legmagasabb helyiértékű bitje.

9 Interrupt-enable (megszakítás-engedélyezés): ha értéke 1, a megszakítások engedélyezettek.

10 Direction (irány): ha 1 az értéke, akkor a feldolgozó a karaktersorozat (sztring) végénél kezdődik (az SI és DI indexregiszterek tartalma automatikusan csökken), egyébként a feldolgozás a sztring elején kezdődik (SI, DI pedig értéke növekszik).

11 Overflow (túlcsordulás): értéke aritmetikai művelet végrehajtása után a legmagasabb és az azt megelőző helyiértékű biten keletkezett átvitellel végzett kizáró "vagy" művelet eredménye.

3.5.6 Bázis-relatív operandusok

Alakjuk:

eltolás **[BX]**
eltolás **[BP]**

A szögletes zárójel itt nem jelölés, leírása kötelező!

A bázis-relatív operandusok a BX vagy BP bázisregiszterek valamelyikéhez relatív memóriacímeket jelentenek. Az eltolás közvetlen vagy közvetett memória-operandus lehet, amelynek értéke egy szám vagy egy memóriacím. Ha nem adjuk meg az eltolást, akkor 0 az alapértelmezés. A bázis-relatív operandus tényleges címe az adott regiszter tartalmának és az eltolásnak az összege. A BX bázisregiszter használata esetén a kiszámított cím a DS regiszter által kijelölt szegmenshez relatív. Ha a BP regisztert használjuk, akkor a bázis-relatív operandus címe az SS regiszter által kijelölt szegmenshez relatív.

A bázis-relatív operandusokat az eddig leírtaktól eltérő formában is megadhatjuk, pl. [BP].eltolás, vagy [BX + -eltolás], de a címkiszámítási algoritmust ez nem befolyásolja.

3.5.7 Indexelt operandusok

Alakjuk:

eltolás **[DI]**
eltolás **[SI]**

A szögletes zárójel itt nem jelölés, kiírása kötelező!

Az indexelt operandus a DI vagy SI indexregiszterek valamelyikéhez relatív címet jelent. Az eltolás közvetlen vagy közvetett memória-operandus lehet, amelynek értéke egy szám vagy egy memóriacím. Ha nem adjuk meg az eltolást, akkor 0 az alapértelmezés.

Az indexelt operandus tényleges címe az adott regiszter tartalmának és az eltolás értékének az összege. Az így kiszámított cím mindig a DS regiszter által kijelölt szegmenshez relatív.

Az indexelt operandusokat az eddig leírtaktól eltérő formában is megadhatjuk, pl. [DI].eltolás, vagy [SI + -eltolás], de ez nem befolyásolja a címkiszámítási algoritmust.

3.5.8 Bázis-relatív indexelt operandusok

Alakjuk:

eltolás **[BX][DI]**
eltolás **[BX][SI]**
eltolás **[BP][DI]**
eltolás **[BP][SI]**

A szögletes zárójel itt nem jelölés, kiírása kötelező!

A bázis-relatív indexelt operandusok a bázis- és indexregiszter kombinációjához relatív memóriacímeket jelentenek. Az eltolás közvetlen vagy közvetett memória-operandus lehet, amelynek értéke egy szám vagy egy memóriacím. Ha nem adjuk meg az eltolást, akkor 0 az alapértelmezés.

A bázis-relatív indexelt operandus tényleges címe az adott regiszterek tartalmának és az eltolás értékének az összege. Ha a BP regisztert használjuk, akkor a kiszámított cím az SS regiszter által kijelölt szegmenshez relatív, egyéb esetben viszont a DS regiszter által kijelölt szegmenshez.

A bázis-relatív indexelt operandusokat az eddig leírtaktól eltérő formában is meg lehet adni, pl. [eltolás][BP][DI], vagy [BX + SI + -eltolás], de ez a címkiszámítási algoritmust nem befolyásolja.

3.5.9 Struktúra-operandusok

Alakjuk:

változó.mező

A struktúra-operandus egy struktúra valamely elemének memóriacímét jelenti. A *változó* vagy egy struktúra azonosítója, vagy egy memória-operandus, amely egy struktúra címét tartalmazza. A *mező* a struktúra valamely elemének az azonosítója. A struktúrák deklarációját a 5. fejezet 2.4 pontja tartalmazza.

A struktúra-operandus tényleges címe a *változó* és a *mező* relatív címének az összege. A cím ahhoz a szegmenshez vagy csoporthoz relatív, amelyben a változót definiáltuk. A struktúra-operandust gyakran használjuk veremben lévő adatok eléréséhez. Ennek egyik módja az, hogy az aktuális veremcímet a BP regiszterbe tesszük és a verem elemeire a következőképpen hivatkozunk:

[BP].elem

Így a verem minden elemét tetszőleges formában elérhetjük.

3.5.10 Rekord-operandusok

Alakjuk:

rekordnév < [*érték*],,, >

A rekord-operandus egy rekordtípus adataira hivatkozik. Az operandusok kifejezésekben is szerepelhetnek. A *rekordnév* egy olyan rekordtípus azonosítója, amelyet a forrás-fileban definiáltunk. Az opcionális *érték* a rekord valamely mezőjének a tartalma. Több érték megadása esetén az adatokat vesszővel kell elválasztani egymástól. Ha egy mező nincs kitöltve, a fordító annak alapértelmezésével fog dolgozni. A csúcsos zárójelek megadása akkor is kötelező, ha benne egyetlen érték sem szerepel.

A rekordok megadási módját a 5. fejezet 2.5 pontja tartalmazza.

3.5.11 Rekordmező-operandusok

Alakjuk:

rekordmezőazonosító

A rekordmező-operandus a mezőnek a megfelelő rekordban való helyét jelenti. Az operandus értéke a mező legalacsonyabb helyiértékű bitjének a pozíciója. Konstans operandusként használható. A rekordmező azonosítójának egy előzőleg már definiált mező azonosítójának kell lennie.

3.6 Az operandusokból előállítható kifejezések

A kifejezések operandusokból és műveleti jelekből állnak. A kifejezésben felírt műveletek eredményeképpen egy értéket vagy egy memóriacímet kapunk attól függően, hogy milyen típusú operandusokat és műveleti jeleket használtunk. A műveleti jelek típusai:

- aritmetikai
- léptető
- összehasonlító
- logikai
- attributum-kezelő

3.6.1 Aritmetikai műveletek

A műveletek jelölése:

kifejezés1 * *kifejezés2*
kifejezés1 / *kifejezés2*
kifejezés1 **MOD** *kifejezés2*
kifejezés1 + *kifejezés2*
kifejezés1 - *kifejezés2*
+*kifejezés*
-*kifejezés*

A műveleti jelek jelentése:

két operandusú műveletek:

*	szorzás
/	osztás
MOD	osztás utáni maradék (modulus)
+	összeadás
-	kivonás

egy operandusú műveletek:

+	pozitív
-	negatív

Aritmetikai műveletet - az összeadás és a kivonás kivételével - csak egész számokkal lehet végezni. Az összeadás műveletével egy relokálható memória-operandushoz hozzáadhatunk egy egész számot. Lehetőség van egy relokálható operandusból egész szám kivonására, vagy két relokálható operandus különbségének kiszámítására is. Ez utóbbi csak akkor lehetséges, ha a relokálható operandusok azonos szegmensben helyezkednek el.

Példák:

3 * 2	= 6
3 / 2	= 1
3 MOD 2	= 1

```
3 + 2          = 5
3 - 2          = 1
3 - +2         = 1
3 - -2         = 5
a + 2
a - 2
a - b
```

3.6.2 Léptető műveletek

Felírási módjuk:

kifejezés **SHR** *lépésszám*
kifejezés **SHL** *lépésszám*

Az SHR művelettel annyi bittel léptetjük jobbra az adott kifejezést, amennyit lépésszámként megadtunk. SHL-lel ugyanígy balra lehet léptetni. A kilépő bitek elvesznek. Ha a lépésszám > = 16, akkor az eredmény 0 lesz.

Példák:

```
10101010B SHR 5 = 00000101B
10101010B SHL 5 = 01000000B
```

3.6.3 Logikai műveletek

Felírási módjuk:

kifejezés1 **EQ** *kifejezés2*
kifejezés1 **NE** *kifejezés2*
kifejezés1 **LT** *kifejezés2*
kifejezés1 **LE** *kifejezés2*
kifejezés1 **GT** *kifejezés2*
kifejezés1 **GE** *kifejezés2*
NOT *kifejezés*
kifejezés1 **AND** *kifejezés2*
kifejezés1 **OR** *kifejezés2*
kifejezés1 **XOR** *kifejezés2*

Az összehasonlító műveletekkel két olyan kifejezést hasonlíthatunk össze, amelyek konkrét értékkel rendelkeznek. Az összehasonlító művelet eredménye igaz logikai érték (0FFFFH), ha a feltétel teljesül, ellenkező esetben hamis logikai érték (0000H). Az összehasonlító műveleteket a fordítás vezérlésére használjuk feltételes direktívákban és feltételes utasításokban. Az összehasonlító műveletek jelentése:

EQ	**kifejezés1 = kifejezés2**
NE	**kifejezés1 < > kifejezés2**
LT	**kifejezés1 < kifejezés2**
LE	**kifejezés1 < = kifejezés2**

GT kifejezés1 > kifejezés2
GE kifejezés1 > =kifejezés2

Példák:

20 EQ 10 hamis
20 NE 10 igaz
20 LT 10 hamis
20 LE 10 hamis
20 GT 10 igaz
20 GE 10 igaz

A logikai műveletekben szereplő kifejezéseknek konkrét értékkel kell rendelkezniük. A kifejezések tetszőleges típusúak. Hosszúságuk attól függ, milyen utasításban használjuk a logikai műveletet. Az egyes műveleti jelek jelentése:

NOT negáció
AND logikai "és"
OR logikai "vagy"
XOR logikai "kizáró vagy"

A logikai műveletek bitenként hajtódnak végre.

Példák:

NOT 00000001B = 11111110B
11110000B AND 00001111B = 00000000B
11110000B OR 00001111B = 11111111B
11110000B XOR 00011111B = 11101111B

3.6.4 Indexelés

Felírási módja:

kifejezés1[*kifejezés2*]

A felírási módban ismét nem tudtuk jelölésrendszerünket alkalmazni. "Kifejezés1" opcionális, "kifejezés2"-t viszont kötelező megadni és *szögletes zárójelpár közé tenni, mivel itt a szögletes zárójelpár az indexelés műveleti jele.*

Az indexelést általában egy tömb vagy egy karaktersorozat elemeire való hivatkozáskor használjuk. Az indexelés művelete megegyezik az összeadás műveletével, kivéve, ha a "kifejezés1"et nem adjuk meg. Ha "kifejezés1"-et elhagyjuk, akkor az indexelési művelet eredményeképpen az egész típusú 0 értéket kapjuk.

"Kifejezés1" egész szám, abszolút szimbólum vagy relokálható operandus lehet. Ha "kifejezés1" nem szerepel, "kifejezés2"-nek egész számnak vagy abszolút szimbólumnak kell lennie.

Egyéb esetben "kifejezés2" is egész szám, abszolút szimbólum vagy relokálható operandus lehet.

Példák:

tömb[6]
string[2]

3.6.5 A PTR művelet

Felírási módja:

típus **PTR** *kifejezés*

A PTR műveletet előrehivatkozáskor használjuk azért, hogy így explicit módon adjuk meg a hivatkozott objektum méretét vagy távolságát. Előrehivatkozás alatt azt értjük, hogy azt a szimbólumot, amelyre hivatkozunk, csak később definiáljuk a programban. Ha előrehivatkozáskor nem alkalmazzuk a PTR műveletet, a fordító az objektum típusára vonatkozó alapfeltételezést használja, és ez hibajelzéshez vezethet, ha az alapfeltételezés nem felel meg az objektum későbbi definíciójának.

A PTR művelettel a kifejezésben megadott változó vagy címke típusát határozhatjuk meg. A "típus" a következő kulcsszavak vagy az azokat jelölő értékek valamelyike lehet:

BYTE	1
WORD	2
DWORD	4
QWORD	8
TBYTE	10
NEAR	0FFFFH
FAR	0FFFEH

A *"kifejezés"* tetszőleges operandus lehet. A BYTE, WORD és DWORD típusokat csak memória-operandusokra lehet megadni. A NEAR és FAR típus csak címkékre használható.

Példa:

NEAR PTR *címke1*
FAR PTR *címke2*

NEAR típusú címkére csak abban a szegmensben lehet hivatkozni, amelyben azt definiáltuk. FAR típusú címkére a program bármely szegmensében hivatkozhatunk. (Az adatok és címkék típusára vonatkozóan a IV. fejezet 2.1 és 2.2 pontjában található bővebb információ.)

3.6.6 A címkiszámítás viszonyítási alapjának megváltoztatása

(jele a ":" karakter)

Felírási módjai:

szegmensregiszter : *kifejezés*
szegmensnév : *kifejezés*
csoportnév : *kifejezés*

Ha ilyen műveletet adunk meg, akkor a fordító a címke vagy változó címének kiszámításához az alapértelmezés helyett az adott szegmensregiszter, szegmensnév vagy csoportnév kezdetét használja. Ha szegmensnevet vagy csoportnevet adunk meg, akkor ezeknek olyan azonosítóknak kell lenniük, amelyeket előzőleg egy ASSUME direktívával egy szegmensregiszterhez rendeltünk és SEGMENT vagy GROUP direktívával definiáltunk. A "kifejezés" abszolút szimbólum vagy relokálható operandus lehet. Szegmensregiszterként a CS, DS, SS és ES regiszterek valamelyike szerepelhet.

A címkiszámítás alapértelmezése címkékre NEAR, a memóriaoperandusokra pedig - az utasítás és az operandus típusától függően - a DS, SS vagy ES regiszterhez relatív.

Példa:

ES:[BX][SI]

Az objektum címe az ES szegmensregiszterhez relatív. A relatív cím a BX bázisregiszter tartalmának és az SI indexregiszter tartalmának az összege. (Az ASSUME, SEGMENT és GROUP direktívák leírását a IV. fejezet 1. pontja tartalmazza.)

3.6.7 A SHORT művelet

Felírási módja:

SHORT *címke*

A SHORT művelet az adott címke típusát SHORT-ra állítja be. A SHORT címkét használó utasítás 1 byte-tal kevesebb helyet foglal, mint ha ugyanebben az utasításban NEAR típusú címkét adunk meg. A SHORT típusú címkét JMP (jump) utasításban lehet használni, ha a címkének az utasítástól való távolsága nem nagyobb, mint 127 byte.

Példa:

SHORT számok

3.6.8 A THIS művelet

Felírási módja:

THIS *típus*

A THIS művelet egy olyan operandust definiál, amelynek címe megegyezik a számláló aktuális értékével, típusát pedig a következő kulcsszavak valamelyike határozza meg:

NEAR
FAR
BYTE
WORD
DWORD
QWORD
TBYTE

A THIS műveletet az EQU vagy = direktívában használjuk. Hatása ugyanaz, mint a LABEL direktíva használatának.

Például

```
ciklus EQU THIS WORD          és
ciklus LABEL WORD
```

egy olyan szó operandust definiál, amelynek címe a számláló aktuális értéke és "ciklus" az azonosítója. (Az EQU, = és a LABEL direktíva részletes leírása a IV. fejezet 2.3 pontjában található.)

3.6.9 A HIGH és a LOW művelet

Felírási módjuk:

HIGH *kifejezés*
LOW *kifejezés*

A HIGH művelet a kifejezés felső 8 bitjét, a LOW művelet a kifejezés alsó 8 bitjét választja le. A "kifejezés" tetszőleges érték lehet. A művelet a 8-bites macro assemblerekkel való kompatibilitása miatt szerepel, s használata ellenjavalt.

Példák:

```
HIGH 0F0F0H = 11110000B
LOW szó = 00001010B, ha a "szó" tartalma 0ED0AH
```

3.6.10 A SEG művelet

Felírási módja:

SEG *kifejezés*

A SEG művelettel azt a szegmensértéket kapjuk meg, amelyhez a "kifejezés" relatív. A "kifejezés" címke, változó, szegmensnév, csoportnév vagy egyéb szimbólum lehet.

Példa:

```
SEG adatok
```

3.6.11 Az OFFSET művelet

Felírási módja:

OFFSET *kifejezés*

Az OFFSET művelet az adott kifejezés eltolását adja vissza. A "kifejezés" címke, változó, szegmensnév vagy egyéb szimbólum lehet. A visszaadott érték azoknak a byte-oknak a száma, amelyek az objektum és az azt definiáló szegmens kezdete között helyezkednek el. Ha a műveletet szegmensnévre alkalmazzuk, akkor a visszaadott érték a szegmens kezdetének és a szegmens utoljára generált byte-jának távolsága.

Ha az OFFSET műveletet a címkiszámítás viszonyítási alapját megváltoztató művelettel együtt használjuk, akkor a kifejezésben megadott objektumnak az adott szegmens kezdetétől mért, byte-okban való távolságát kapjuk vissza.

A visszakapott érték mindig relatív, a program szerkesztésekor változhat.

Példa:

OFFSET csoport:vektor

Ennek hatására a "vektor" változónak a "csoport" szegmens kezdetétől byte-okban mért távolságát kapjuk vissza.

3.6.12 A TYPE művelet

Felírási módja:

TYPE *kifejezés*

A TYPE művelettel egy olyan értéket kapunk, amely az adott kifejezés típusára jellemző. Ha a kifejezés egy változó, akkor a visszaadott érték az operandus mérete byte-okban. Ha a kifejezés címke, akkor a TYPE művelet eredményeképpen NEAR típusú címke esetén 0FFFFH-t, FAR típusú címke esetében 0FFFEH-t kapunk. A visszakapott értéket fel lehet használni a típus meghatározására a PTR műveletben.

Példa:

TYPE vektor

3.6.13 A .TYPE művelet

Felírási módja:

.TYPE *kifejezés*

A .TYPE művelet hatására egy olyan byte-ot kapunk vissza, amely az adott kifejezés jellemzőit és hatáskörét tartalmazza. Ha a "kifejezés" nem definiált, akkor a visszakapott byte értéke nulla.

A változó attributumait a byte nulladik, első, ötödik és hetedik bitpozícióján kapjuk meg és jelentésük a következő:

bitpozíció	bit = 0	bit = 1
0	abszolút	programfüggő
1	nem inicializált	inicializált
5	definiálatlan	definiált
7	lokális hatáskörű	external hatáskörű (más modulban definiált)

Ha a hatáskört és a definiáltságot jelző bit értéke 0, akkor a kifejezés érvénytelen.

A .TYPE műveletet általában feltételes direktívákban használjuk, hogy az argumentum attritubumai szerint vezéreljük a programfordítást.

Példa:

```
a DB 54
b EQU .TYPE a
```

Hatására b értéke 34 lesz, mert az "a" nevű változó (= 54) abszolút, inicializált, definiált és lokális hatáskörű.

3.6.14 A LENGTH művelet

Felírási módja:

LENGTH *változó*

A LENGTH művelet hatására az adott változó elemeinek számát kapjuk meg. A mértékegység byte, szó, dupla szó, négy szó vagy tíz byte lehet attól függően, hogy a változót BYTE, WORD, DWORD, QWORD vagy TBYTE típusúra definiáltuk.

A visszakapott érték csak akkor lehet 1-nél nagyobb, ha a változó definíciójában a DUP operátor is szerepel. *A LENGTH művelet hatására ekkor mindig az első DUP operátort megelőző számot kapjuk, tekintet nélkül arra, hogy a változó definíciója tartalmaz-e több DUP operátort is.*

A DUP műveleti jel segítségével vektort vagy tömböt definiálunk úgy, hogy egy vagy több kezdőértéket is ismételhetünk. A következőképpen adható meg:

```
szám DUP(kezdőérték,,,)
```

A "szám" határozza meg, hogy hányszor ismétlődjön a kezdőérték. A kezdőérték lehet kérdőjel karakter (határozatlan kezdőértéket jelent), vagy egy olyan kifejezés, amelynek értéke egy egész szám, karakterkonstans vagy egy másik DUP művelet. A DUP műveletek 17 szint mélységig ágyazhatók egymásba. Ha egynél több kezdőértéket adunk meg, akkor a kezdőértékek sorozatonként ismétlődnek.

Például a

```
tömb DW 4 DUP(3 DUP(2 DUP(1,4,5)))
```

direktíva 24 szót foglal le és a lefoglalt szavak tartalma: 1,4,5,1,4,5,1,4,5,...

Ekkor a

```
LENGTH tömb
```

művelet eredménye 4 lesz, a

```
SIZE tömb
```

eredménye viszont 48.

3.6.15 A SIZE művelet

Felírási módja:

SIZE *változó*

A SIZE művelettel az adott változó számára lefoglalt terület byte-okban kifejezett méretét lehet lekérdezni.

3.6.16 A WIDTH művelet

Felírási módja:

WIDTH *rekordmező-azonosító | rekordazonosító*

A WIDTH művelettel az adott rekordmező vagy rekord bitekben mért hosszát kérdezhetjük le. Ha például a "rek" rekordot a következőképpen definiáltuk:

```
rtip RECORD a:2, b:5, c:1
rek  rtip  < >
```

akkor

```
WIDTH a    = 2
WIDTH b    = 5
WIDTH c    = 1
WIDTH rtip = 8
```

3.6.17 A MASK művelet

Felírási módja:

MASK *rekordmező-azonosító | rekordazonosító*

A MASK művelet hatására a rekord vagy a rekordban előforduló mezők bitmaszkját kaphatjuk meg. A maszkban azok a bitek lesznek 1 értékűek, amelyeknek megfelelő bitek az adott mezőhöz tartoznak. A rekord bitmaszkja 0FFH vagy 0FFFFH attól függően, hogy a rekord 8-bites vagy 16 bit hosszú.

Ha például a "rek" rekordot a következőképpen definiáltuk:

```
rtip RECORD a:2, b:5, c:1
rek  stip  < >
```

ami azt jelenti, hogy az "rtip" rekordtípus 8-bites, amelynek felépítése:

```
   a   |       b       | c |      mező
 7 6   | 5 4 3 2 1     | 0 |   bitpozició
```

és az "rtip" típusú "rek" rekordnak nincs kezdőértéke. Ekkor

MASK a = 11000000B = 0C0H

```
MASK b      = 00111110B = 3FH
MASK c      = 00000001B = 01H
MASK rtip   = 11111111B = 0FFH
```

3.6.18 A kifejezés értékének kiszámítása és a precedencia

A kifejezés értékének kiszámítása a műveleti jelek precedenciaszabályai és felírási sorrendje szerint történik. Elsőként a legmagasabb precedenciájú műveletek hajtódnak végre. *Az azonos precedenciájú műveletek elvégzése a felírási sorrendben, balról jobbra történik.* **A műveletek elvégzésének ezt a sorrendjét zárójelezéssel meg lehet változtatni.** A zárójelben lévő műveletek mindig előbb kerülnek végrehajtásra, mint a szomszédos műveletek.

A következő táblázat a műveleti jelek precedenciaszabályait foglalja össze. Az azonos sorban lévő műveletek azonos precedenciájúak. A legmagasabb precedenciájú műveletek az 1-es sorban lévők.

Precedencia	**művelet**
1	LEGTH, SIZE, WIDTH, MASK
2	()
3	[]
4	:
5	PTR, OFFSET, SEG, TYPE, THIS
6	HIGH, LOW
7	*, /, MOD, SHL, SHR
8	+, -
9	EQ, NE, LT, LE, GT, GE
10	NOT
11	AND
12	OR, XOR
13	SHORT, .TYPE

3.6.19 A memória-operandusok típusának ellenőrzése

A makro-assembler szigorú szintaktikus ellenőrzést végez az utasításokon, beleértve azokat az operandusokat, amelyek memóriaterületekre hivatkoznak. Ez azt jelenti, hogy az implicit adattípusú relokálható operandusokkal megadott utasításokban az operandusoknak vagy azonos típusúaknak kell lenniük, vagy explicit módon, a PTR művelettel azonos típusúakká kell tenni azokat. Ha ezt elmulasztjuk, a fordító hibajelzést ad.

4. A mikroprocesszor-család assembly utasításai

Ebben a fejezetben összefoglaljuk a mikroprocesszor-család assembly utasításkészletét. Az utasítások működésének lényegét az első fejezetben már ismertettük. Részletes magyarázatok és a címzési módok leírása rendelkezésre áll már a magyar számítástechnikai szakirodalomban (pl. az LSI ATSZ kiadásában: AZ i8086 mikroprocesszor), ezért itt csak az utasítások szintaktikáját és rövid jelentését soroljuk fel.

4.1 Az utasítások leírásában használt rövidítések

Rövidítés	Értelmezés
akku	akkumulátor: **AX** vagy **AL** regiszter
reg	egy byte vagy egy szó hosszúságú regiszter byte: **AL, AH, BL, BH, CL, CH, DL, DH,** szó: **AX, BX, CX, DX, SI, DI, BP, SP**
szreg	szegmensregiszter: **CS, DS, SS, ES**
ált	általános operandus: regiszter, memóriacím, indexelt operandus, bázis-relatív operandus vagy bázis-relatív indexelt operandus
konst	8- vagy 16-bites konstans vagy literál
mem	memória-operandus: címke, változó vagy literál
címke	utasításcímke
IP	utasítás-számláló (16 bit)
SP	verem-mutató (16 bit)
BP	bázis-regiszter (16 bit)
(...)	a zárójelben megadott regiszter vagy memóriaterület tartalma
cím + 1:cím	a "cím"-en kezdődő, két egymást követő byte a memóriában
%	maradékképzés
&	logikai "és"
IOR	logikai "vagy" (megengedő)
XOR	logikai "kizáró vagy"
flag	16 bites regiszter, amelyben a jelzőbitek (flag-ek) találhatók
CF	átviteli (carry) flag
PF	paritás (parity) flag
AF	másodlagos átviteli (auxilary carry) flag
ZF	nulla (zero) flag
SF	előjel (sign) flag
TF	trap flag
IF	megszakítási (interrupt) flag
DF	irányt jelző (direction) flag
OF	túlcsordulási (overflow) flag

4.2 A 8086 és a 8088 utasításkészlete

AAA

Két négy bites nem tömörített (felső 4 bit nulla) BCD szám összeadása után, ha az eredmény AL-ben van, AL értékét átalakítja nem tömörített BCD számmá (számokká).

A processzor lépései:

ha ((AL) & 0FH) > 9 vagy (AF) = 1 akkor
(AL) = (AL) + 6
(AH) = (AL) + 1
(AF) = 1
(CF) = (AF)
(AL) = (AL)&0FH

Állított flag-ek: AF, CF

Példa: AAA ; igazítás összeadás után

AAD

Az AX regiszterben lévő nem tömörített, binárisan kódolt decimális (BCD) szám konvertálása bináris számmá. Az eredmény az AL regiszterbe kerül. Osztás előtt az osztandó konvertálására szokták használni.

A processzor lépései:

(AL) = (AH)*0AH + (AL)
(AH) = 0

Állított flag-ek: PF, SF, ZF

Példa: AAD ; binárissá konvertálás osztás előtt

AAM

Nem tömörített,előjel nélküli BCD operandusokkal végzett szorzás AL-ben keletkező bináris eredményének konvertálása binárisan kódolt decimális számmá. Az eredmény az AX regiszterbe kerül.

A processzor lépései:

(AH) = (AL)/0AH
(AL) = (AL)%0AH %: maradék

Állított flag-ek: PF, SF, ZF

Példa: AAM ;BCD-vé konvertálás szorzás után

AAS

Nem tömörített BCD operandusokkal végzett kivonás AL regiszterben keletkező eredményének nem tömörített átalakítása BCD számmá.

A processzor lépései:

ha ((AL) & 0FH) > 9 vagy (AF) = 1 akkor
(AL) = (AL)-6
(AH) = (AH)-1
(AF) = 1
(CF) = (AF)
(AL) = (AL)&0FH

Állított flag-ek: AF,CF

Példa: AAS ; igazítás kivonás után

ADC *akku,konst*

Konstans vagy literál hozzáadása az akkumulátor tartalmához az átviteli (carry) flag állításával.

A processzor lépései:

ha (CF) = 1 akkor (akku) = (akku) + konst + 1
ha (CF) = 0 akkor (akku) = (akku) + konst

Állított flag-ek: AF, CF, OF, SF, ZF

Példa: ADC AX,123

ADC *ált,konst*

Konstans vagy literál hozzáadása egy általános operandushoz az átviteli flag állításával.

A processzor lépései:

ha (CF) = 1 akkor (ált) = (ált) + konst + 1
ha (CF) = 0 akkor (ált) = (ált) + konst

Állított flag-ek: AF, CF, OF, PF, SF, ZF

Példa: ADC mem-cím,321

ADC *ált,reg*

Regiszter tartalmának hozzáadása egy általános operandushoz az átviteli flag állításával.

A processzor lépései:

ha (CF) = 1 akkor (ált) = (ált) + (reg) + 1
ha (CF) = 0 akkor (ált) = (ált) + (reg)

Állított flag-ek: AF, CF, OF, PF, SF, ZF

Példa: ADC mem-cím,CX

ADC *reg,ált*

Általános operandus hozzáadása egy regiszter tartalmához az átviteli flag állításával.

A processzor lépései:

ha (CF) = 1 akkor (reg) = (reg) + (ált) + 1
ha (CF) = 0 akkor (reg) = (reg) + (ált)

Állított flag-ek: AF, CF, OF, PF, SF, ZF

Példa: ADC CX,mem-cím

ADD *akku,konst*

Konstans vagy literál hozzáadása az akkumulátor tartalmához.

A processzor lépése:

(akku) = (akku) + konst

Állított flag-ek: AF, CF, OF, PF, SF, ZF

Példa: ADD AL,2

ADD *ált,konst*

Konstans vagy literál hozzáadása egy általános operandushoz.

A processzor lépése:

(ált) = (ált) + konst

Állított flag-ek: AF, CF, OF, PF, SF, ZF

Példa: ADD mem-cím,34

ADD *ált,reg*

Regiszter tartalmának hozzáadása egy általános operandushoz.

A processzor lépése:

(ált) = (ált) + (reg)

Állított flag-ek: AF, CF, OF, PF,SF, ZF

Példa: ADD mem[BP][DI],BX

ADD *reg,ált*

Általános operandus hozzáadása egy regiszter tartalmához.

A processzor lépése:

(reg) = (reg) + (ált)

Állított flag-ek: AF, CF, OF, PF,SF, ZF

Példa: ADD DX,szó[BX][DI]

AND *akku,konst*

Logikai "és" művelet az akkumulátor tartalma és egy konstans vagy literál között. Az eredmény az akkumulátorba kerül.

A processzor lépései:

(akku) = (akku)&konst
(CF) = 0
(OF) = 0

Állított flag-ek: CF, OF, PF, SF, ZF

Példa: AND AX,2B4AH

AND *ált,konst*

Logikai "és" művelet egy általános operandus és egy konstans vagy literál között. Az eredmény felülírja az általános operandus eredeti értékét.

A processzor lépései:

(ált) = (ált)&konst
(CF) = 0
(OF) = 0

Állított flag-ek: CF, OF, PF, SF, ZF

Példa: AND mem-byte,23H

AND *ált,reg*

Logikai "és"művelet egy általános operandus és egy regiszter tartalma között. Az eredmény felülírja az általános operandus eredeti értékét.

A processzor lépései:

(ált) = (ált)&(reg)
(CF) = 0
(OF) = 0

Állított flag-ek: CF, OF, PF, SF, ZF

Példa: AND mem-szó,BP

AND *reg,ált*

Logikai "és" művelet egy regiszter tartalma és egy általános operandus között. Az eredmény a regiszterbe kerül.

A processzor lépései:

(reg) = (reg)&(ált)
(CF) = 0
(OF) = 0

Állított flag-ek: CF, OF, PF, SF, ZF

Példa: AND BX,mem-szó

CALL *címke*

A "címke" által meghatározott szubrutin hívása. A címke a hívó szegmensen belül vagy egy másik szegmensben definiált. Szegmensen belüli híváskor:IP verembe kerül, a hívási cím = IP + előjeles eltolási cím. Szegmensek közötti híváskor először a CS majd az IP kerül a verembe; a hívásnak megfelelően beáll az új CS és IP érték.

A processzor lépései:

1. ha a címke a hívó szegmensen kívül van, akkor
   (SP) = (SP)-2
   ((SP) + 1:(SP)) = (CS)
   (CS) = SEG
2. (SP) = (SP)-2
3. ((SP) + 1:(SP)) = (IP)
4. (IP) = DEST

Hívó szegmensen belüli címke esetén:

DEST = (IP) + disp (disp: a címke távolsága a CALL utasítás végétől)

Hívó szegmensen kívüli címke esetén:

DEST = offset, SEG = szegmensszám

Állított flag-ek: -

Példák:
CALL near-label
CALL near-proc
CALL far-label
CALL far-proc

CALL *ált*

Indirekt szubrutinhívás. A hívott rutin a hívó szegmensen belül vagy egy másik szegmensben definiált.

A processzor lépései ugyanazok, mint direkt szubrutinhívás esetén, de a következő behelyettesítéseket kell figyelembe venni:

Ha a rutin a hívó szegmensen kívül van:

DEST = (ált), SEG = (ált + 2)

Ha a rutin a hívó szegmensen belül van:

DEST = (ált)

CBW

Byte konvertálása szóvá: az AL regiszter 7.bitjének értékével töltödik fel az AH regiszter

minden bitje. Egy byte-os előjeles számok 2 byte-os előjeles számmá való átalakítása, egy byte-os előjeles osztási műveletek előtt.

Állított flag-ek: -

CLC

Az átviteli flag (carry) törlése.

A processzor lépése:

(CF) = 0

Állított flag: CF

CLD

Az irányt jelző flag törlése. Az SI és/vagy DI regisztereket változtató ciklus utasítások, ha a direction flag nulla (CLD utasítás után) az SI,DI regisztereket eggyel növelik. Ha a flag értéke 1, akkor eggyel csökkentik.

A processzor lépése:

(DF) = 0

Állított flag: DF

CLI

Hatására a maszkolható hardware megszakítások letiltásra kerülnek, az interrupt flag értéke nulla. Az utasítás a software interrupt-ra és az NMI-re nincs hatással.

A processzor lépése:

(IF) = 0

Állított flag: IF

CMC

Az átviteli (carry) flag átállítása az eredeti érték ellentettjére.

A processzor lépése:

ha (CF) = 1-(CF)

Állított flag: CF

CMP *akku,konst*

Az akkumulátor tartalmának összehasonlítása egy konstanssal vagy literállal. Az operandusok értéke nem változik. Az utasítás tulajdonképpen kivonja az akku értékét a konst-ból, az eredmény azonban nem jelenik meg, csupán az eredménynek megfelelő jelzőbitek állnak be.

A processzor lépése:

(akku)-konst

Állított flag-ek: AF, CF, OF, PF, SF, ZF

Példa: CMP AL,4

CMP *ált,konst*

Általános operandus összehasonlítása egy konstanssal vagy literállal. Az operandusok értéke nem változik.

A processzor lépése:

(ált)-konst

Állított flag-ek: AF, CF, OF, PF, SF, ZF

Példa: CMP mem-szó,112

CMP *ált,reg*

Általános operandus összehasonlítása egy regiszter tartalmával. Az operandusok értéke nem változik.

A processzor lépése:

(ált)-(reg)

Állított flag-ek: AF, CF, OF, PF, SF, ZF

Példa: CMP mem-szó,BX

CMP *reg,ált*

Regiszter tartalmának összehasonlítása egy általános operandussal. Az operandusok értéke nem változik.

A processzor lépése:

(reg)-(ált)

Állított flag-ek: AF, CF, OF, PF, SF, ZF

Példa: CMP CX,tár[BP][SI]

CMPS *string1*, *string2* vagy **CMPSB** vagy **CMPSW**

Karaktersorozatok összehasonlítása. Az operandusok értéke változatlan marad. String1 címe DS:SI-ben, String2 címe ES:DI-ben van. A flagek itt is beállnak.

A processzor lépései:

(LSRC)-(RSRC)
ha (DF) = 0 akkor
(SI) = (SI) + DELTA

(DI) = (DI) + DELTA
egyébként
(SI) = (SI)-DELTA
(DI) = (DI)-DELTA

Byte-os műveletek esetében:

LSRC = (SI), RSRC = (DI), DELTA = 1

Szavas műveletek esetében:

LSRC = (SI) + 1:(SI), RSRC = (DI) + 1:(DI), DELTA = 2

Példa: MOV SI, OFFSET string1
MOV DI, OFFSET string2
CMPS string1,string2 vagy
CMPS DS:BYTE PTR[SI],ES:[DI] vagy
CMPSB

CWD

Szó konvertálása dupla szóvá: az AX regiszter 15. bitjének értékével töltődik fel a DX regiszter minden bitje. Előjeles, két byte-os osztások előtt használjuk.

A processzor lépése:

ha (AX) < 8000h akkor (DX) = 0 egyébként (DX) = FFFFH

Állított flag: -

DAA

AL regiszter tartalmát tömörített BCD számokká alakítja, BCD számok összeadása után.

A processzor lépései:

ha (AL)&0FH > 9 vagy (AF) = 1 akkor
(AL) = (AL) + 6
(AF) = 1
ha (AL) > 9FH vagy (CF) = 1 akkor
(AL) = (AL) + 60H
(CF) = 1

Állított flag-ek: AF, CF, PF, SF, ZF

DAS

AL regiszter tartalmát tömörített BCD számokká alakítja BCD számok kivonása után.

A processzor lépései:

ha (AL)&0FH > 9 vagy (AF) = 1 akkor
(AL) = (AL)-6
(AF) = 1

ha (AL) > 9FH vagy (CF) = 1 akkor
(AL) = (AL)-60H
(CF) = 1

Állított flag-ek: AF, CF, PF, SF, ZF

DEC *ált*

Egy általános operandus értékének csökkentése 1-el.

A processzor lépése:

(ált) = (ált)-1

Állított flag-ek: AF, OF, PF, SF, ZF

Példa: DEC mem-szó

DIV *ált*

Előjel nélküli bináris számok osztása. 2 byte-os osztáskor az osztandót a DX,AX regiszterpár tartalmazza, amelyet az utasításban megadott operandussal osztunk. A hányados AX-be, a maradék DX-be kerül. Byte-os osztáskor az osztandót AX tartalmazza, az osztó az utasítás operandusa. A hányados AL-be, a maradék AH-ba kerül.

A processzor lépései:

(temp) = (NUMR)
ha (temp)/(DIVR) > MAX akkor
(QUO),(REM) definiálatlan
(SP) = (SP)-2
((SP) + 1:(SP)) = flag-ek
(IF) = 0
(TF) = 0
(SP) = (SP)-2
((SP) + 1:(SP)) = (CS)
(CS) = (2) ; a 2-es es 3-as hely tartalma
(SP) = (SP)-2
((SP) + 1:(SP)) = (IP)
(IP) = (0) ; a 0-ás és 1-es hely tartalma (INT 0 generálódik)
egyébként
(QUO) = (temp)/(DIVR)
(REM) = (temp)%(DIVR)

Byte-os osztás esetén:
NUMR = AX, DIVR = ált, QOU = AL,
REM = AH, MAX = FFH

Szavas osztás esetén:
NUMR = DX:AX, DIVR = ált,QUO = AX,
REM = DX, MAX = FFFFH

Állított flag-ek: nincs érvényes flag

Példa: MOV AX,osztandó
DIV osztó

ESC *konst,ált*

Az utasítás a más processzorokkal (pl:matematikai koprocesszor) való kapcsolattartást biztosítja: a megcímzett tárterület tartalmát az adatbuszra teszi, de nem végez el vele műveletet. Az ESC mnemonik-ot követi egy opkód és egy utasítás is a koprocesszor számára.

HLT

Az utasítás hatására a programfutás felfüggesztődik. A processzor nem maszkolható megszakítás vagy reset hatására folytatja a program végrehajtását.

IDIV *ált*

Előjeles bináris számok osztása. Két byte-os osztáskor az osztandót a DX,AX regiszterpár tartalmazza, amelyet az utasításban megadott operandussal osztunk. A hányados AX-be, a maradék DX-be kerül. Negatív osztandó esetén használjuk előtte a CWD utasítást. Byte-os osztáskor az osztandót az AX tartalmazza, az osztó az utasítás operandusa. A hányados AL-be, a maradék AH-ba kerül. Negatív osztandó esetén használjuk előtte a CBW utasítást.

A processzor lépései:

(temp) = (NUMR)
ha (temp)/(DIVR) > 0 és (temp)/(DIVR) > MAX vagy
(temp)/(DIVR) < 0 és (temp)/(DIVR) < 0-MAX-1 akkor
(QUO),(REM) definiálatlan
(SP) = (SP)-2
((SP) + 1:(SP)) = flag-ek
(IF) = 0
(TF) = 0
(SP) = (SP)-2
((SP) + 1:(SP)) = (CS)
(CS) = 2
(SP) = (SP)-2
((SP) + 1:(SP)) = (IP)
(IP) = (0) (INT 0 generálódik!)egyébként
(QUO) = (temp)/(DIVR)
(REM) = (temp)%(DIVR)

Byte-os osztás esetén:
NUMR = AX, DIVR = ált,QUO = AL,
REM = AH,MAX = FFH

Szavas osztás esetén:

NUMR = DX:AX,DIVR = ált,QUO = AX,
REM = DX,MAX = FFFFH.

Állított flag-ek: AF, CF, OF, PF, SF, ZF

Példa: MOV AX,osztandó[BX]
IDIV osztó

IMUL *ált*

Előjeles bináris számok szorzása. Szavas szorzáskor az egyik tényezőt az AX regiszter tartalmazza, a másik tényező az utasítás operandusa. A szorzat a DX,AX regiszterpárba kerül. Byte-os szorzáskor az egyik tényezőt AL tartalmazza, a másikat az utasítás operandusa. A szorzat AX-be kerül.

A processzor lépései:

(DEST) = (LSRC)*(RSRC)
ha (EXT) = (LOW) előjelkiterjesztése, akkor (CF) = 0
egyébként
(CF) = 1
(OF) = (CF)

Byte-os szorzás esetén:

LSRC = AL, RSRC = ált, DEST = AX,
EXT = AH, LOW = AL

Szavas szorzás esetén:

LSRC = AX, RSRC = ált, DEST = DX:AX,
EXT = DX, LOW = AX

Állított flag-ek: CF, OF

Példa: MOV AX,tényező1-szó
IMUL tényező2-szó

IN *akku,DX*

Byte vagy szó beolvasása AL-be vagy AX-be a DX regiszter által meghatározott I/O portról.

A processzor lépése:

(akku) = ((DX)port)

Állított flag-ek: -

Példa: IN AX,DX

IN *akku,konst*

Byte vagy szó beolvasása AL-be vagy AX-be a konstans vagy literál által megadottcímű i/o portról.

A processzor lépése:

(akku) = (konst port)

Állított flag-ek: -

Példa: IN AL,port-byte

INC *ált*

Az operandus értékének 1-el való növelése.

A processzor lépése:

(ált) = (ált) + 1

Állított flag-ek: AF, OF, PF, SF, ZF

Példa: INC mem-szó[BX]

INT 3

1 byte-os speciális software interrupt.

A processzor lépései:

(SP) = (SP)-2
((SP) + 1:(SP)) = flag-ek
(IF) = 0
(TF) = 0
(SP) = (SP)-2
((SP) + 1:(SP)) = (CS)
(CS) = (3*4 + 2)
(SP) = (SP)-2
((SP) + 1:(SP)) = (IP)
(IP) = (3*4)

Állított flag-ek: IF, TF

Példa: INT 3 ; egy byte: (11001100)

INT *konst*

Software interrupt 0-255 típuskóddal. A 0-255 közötti értékek az ugrási cím vektor táblájára mutatnak. A tábla effektív címe: 00000H-003FFH. Az operációs rendszer kezeli.

A processzor lépései:

(SP) = (SP)-2
((SP) + 1:(SP)) = flag-ek
(IF) = 0
(TF) = 0
(SP) = (SP)-2
((SP) + 1:(SP)) = (CS)

(CS) = (konst*4 + 2)
(SP) = (SP)-2
((SP) + 1:(SP)) = (IP)
(IP) = (konst*4)

Állított flag-ek: IF, TF

Példa: INT 44 ; két byte: (11001101 00101100)

INTO

Software interrupt, ha az overflow flag értéke 1.

A processzor lépései:

ha (OF) = 1 akkor
(SP) = (SP)-2
((SP) + 1:(SP)) = flag-ek
(IF) = 0
(TF) = 0
(SP) = (SP)-2
((SP) + 1:(SP)) = (CS)
(CS) = (12H)
(SP) = (SP)-2
((SP) + 1:(SP)) = (IP)

Állított flag-ek: IF,TF ha túlcsordulás volt

INTR

Maszkolható külső interrupt. Csak a 8088 mikroprocesszor ismeri

IRET

Visszatérés interrupt-ból. IP és CS értéke és a flag-ek visszatöltődnek a veremből. (Az INT utasítások CS és IP aktuális értékét a verembe mentik, utána ugranak el a vektor-tábla által meghatározott címre.)

A processzor lépései:

(IP) = ((SP) + 1:(SP))
(SP) = (SP) + 2
(CS) = ((SP) + 1:(SP))
(SP) = (SP) + 2
flag-ek = ((SP) + 1:(SP))
(SP) = (SP) + 2

Állított flag-ek: minden flag

JA *címke* vagy **JNBE** *címke*

Ugrás a címkére ha a carry flag és a zero flag értéke nulla. A vezérlést csak 128 byte-nal

kisebb távolságra lévő címkére lehet átadni. Az ugrás előjeles.

A processzor lépése:

ha (CF) IOR (ZF) = 0 akkor
(IP) = (IP) + disp (előjelkiterjesztéssel 16 bitre)

ahol "disp" az utasítás operandusaként megadott címkének az utasítás végétől való távolsága.

Állított flag-ek: -

Példa: JA közeli-címke
JNBE közeli-címke

JAE *címke* vagy **JNB** *címke*

Ugrás a címkére, ha a carry flag értéke nulla. A vezérlést csak 128 byte-nál kisebb távolságra lévő címkére lehet átadni. Előjeles ugrás.

A processzor lépése:

ha (CF) = 0 akkor
(IP) = (IP) + disp (előjelkiterjesztéssel 16 bitre)

ahol "disp" az utasítás operandusaként megadott címkének az utasítás végétől való távolsága.

Állított flag-ek: -

Példa: JNB közeli-címke
JAE közeli-címke

JB *címke* vagy **JNAE** *címke* vagy **JC** *címke*

Ugrás a címkére, ha a carry flag értéke 1. A vezérlést csak 128 byte-nál kisebb távolságra lévő címkére lehet átadni. Előjeles ugrás.

A processzor lépése:

ha (CF) = 1 akkor
(IP) = (IP) + disp (előjelkiterjesztéssel 16 bitre)

ahol "disp" az utasítás operandusaként megadott címkének az utasítás végétől való távolsága.

Állított flag-ek: -

Példa: JB közeli-címke
JNAE közeli-címke
JC közeli-címke

JBE *címke* vagy **JNA** *címke*

Ugrás a címkére ha a carry vagy a zero flag értéke 1. A vezérlést csak 128 byte-nál kisebb

távolságra lévő címkére lehet átadni. Előjeles ugrás.

A processzor lépése:

ha (CF) IOR (ZF) = 1 akkor
(IP) = (IP) + disp (előjelkiterjesztéssel 16 bitre)

ahol "disp" az utasítás operandusaként megadott címkének az utasítás végétől való távolsága.

Állított flag-ek: -

Példa: JBE közeli-címke
JNA közeli címke

JCXZ *címke*

Ugrás a címkére, ha a CX regiszter tartalma nulla. A vezérlést csak 128 byte-nál kisebb távolságra lévő címkére lehet átadni. Előjeles ugrás.

A processzor lépése:

ha (CX) = 0 akkor
(IP) = (IP) + disp (előjelkiterjesztéssel 16 bitre)

ahol "disp" az utasítás operandusaként megadott címkének az utasítás végétől való távolsága.

Állított flag-ek: -

Példa: JCXZ közeli címke

JE *címke* vagy **JZ** *címke*

Ugrás a címkére, ha a zero flag értéke 1. A vezérlést csak 128 byte-nál kisebb távolságra lévő címkére lehet átadni. Előjeles ugrás.

A processzor lépése:

ha (ZF) = 1 akkor
(IP) = (IP) + disp (előjelkiterjesztéssel 16 bitre)

ahol "disp" az utasítás operandusaként megadott címkének az utasítás végétől való távolsága.

Állított flag-ek: -

Példa: JE közeli címke
JZ közeli címke

JG *címke* vagy **JNLE** *címke*

Ugrás a címkére, ha az összehasonlítás eredménye "nagyobb". A vezérlést csak 128 byte-nál kisebb távolságra lévő címkére lehet átadni. Az ugrás jelző feltétele: ((előjel flag XOR owerflow flag) AND zero flag) = 0. Előjeles ugrás.

A processzor lépése:

ha ((SF) XOR (OF)) vagy (ZF) = 0 akkor
(IP) = (IP) + disp (előjelkiterjesztéssel 16 bitre

ahol "disp" az utasítás operandusaként megadott címkének az utasítás végétől való távolsága.

Állított flag-ek: -

Példa: JG közeli címke
JNLE közeli címke

JGE *címke* vagy **JNL** *címke*

Ugrás a címkére, ha az összehasonlítás eredménye "nagyobb vagy egyenlő". A vezérlést csak 128 byte-nál kisebb távolságra lévő címkére lehet átadni. Az ugrás jelző feltétele: (előjel flag XOR overflow flag) = 0.Előjeles ugrás.

A processzor lépése:

ha (SF) XOR (OF) = 0 akkor
(IP) = (IP) + disp (előjelkiterjesztéssel 16 bitre)

ahol "disp" az utasítás operandusaként megadott címkének az utasítás végétől való távolsága.

Állított flag-ek: -

Példa: JGE közeli címke
JNL közeli címke

JL *címke* vagy **JNGE** *címke*

Ugrás a címkére, ha az összehasonlítás eredménye "kisebb". A vezérlést csak 128 byte-nál kisebb távolságra lévő címkére lehet átadni. Az ugrás jelző feltétele: (előjel flag XOR overflow flag) = 1. Előjeles ugrás.

A processzor lépése:

ha (SF) XOR (OF) = 1 akkor
(IP) = (IP) + disp (előjelkiterjesztéssel 16 bitre)

ahol "disp" az utasítás operandusaként megadott címkének az utasítás végétől való távolsága.

Állított flag-ek: -

Példa: JL közeli címke
JNGE közeli címke

JLE *címke* vagy **JNG** *címke*

Ugrás a címkére, ha az összehasonlítás eredménye "kisebb vagy egyenlő". A vezérlést csak 128 byte-nál kisebb távolságra lévő címkére lehet átadni. Az ugrás jelző feltétele:

((előjel flag XOR overflow flag)OR zero flag) = 1. Előjeles ugrás.

A processzor lépése:

ha (SF) XOR (OF) IOR (ZF) = 1 akkor
(IP) = (IP) + disp (előjelkiterjesztéssel 16 bitre)

ahol "disp" az utasítás operandusaként megadott címkének az utasítás végétől való távolsága.

Állított flag-ek: -

Példa: JLE közeli címke
JNG közeli címke

JMP *címke*

Feltétel nélküli direkt ugrás a címkére.

A processzor lépése:

ha a címke másik szegmensben van, akkor
(CS) = SEG
(IP) = DEST

Szegmensen belüli vezérlésátadás esetén:
(IP) = (IP) + disp

ahol "disp" az utasítás operandusaként megadott címkének az utasítás végétől való távolsága. Ha ez a távolság 128 byte-nál kisebb, akkor a cím kiszámítása előjel-kiterjesztéssel történik 16 bitre.

Példa: JMP közeli címke

Ha a vezérlést egy másik szegmensben levő címkére akarjuk átadni, akkor

DEST = offset, SEG = szegmenscím

Példa: JMP FAR PTR nyomtat

Állított flag-ek:-

JMP *ált*

Feltétel nélküli indirekt ugrás az operandus által kijelölt címre.

A processzor lépése:

ha a cím másik szegmensben van, akkor
(CS) = SEG
(IP) = DEST

Szegmensen belüli vezérlésátadás esetén
(IP) = (ált)

Példa: JMP szavas-változó

Ha a vezérlést egy másik szegmensben lévő címre akarjuk átadni, akkor

DEST = (ált)
SEG = (ált + 2)

Példa: JMP duplaszavas-változó

JNC *címke*

Ugrás a címkére, ha az átvitel (carry) flag értéke nulla. Előjeles ugrás.

A processzor lépése:

ha (CF) = 0 akkor
(IP) = (IP) + disp (előjelkiterjesztéssel 16 bitre)

Példa: JNC közeli címke

JNE *címke*

Ugrás a címkére, ha a zero flag értéke nulla. A vezérlést csak 128 byte-nál kisebb távolságra lévő címkére lehet átadni. Előjeles ugrás.

A processzor lépése:

ha (ZF) = 0 akkor
(IP) = (IP) + disp (előjelkiterjesztéssel 16 bitre)

ahol "disp" az utasítás operandusaként megadott címkének az utasítás végétől való távolsága.

Állított flag-ek: -

Példa: JNE közeli-címke
JNZ közeli-címke

JNO *címke*

Ugrás a címkére, ha az overflow flag értéke nulla. A vezérlést csak 128 byte-nál kisebb távolságra lévő címkére lehet átadni. Előjeles ugrás.

A processzor lépése:

ha (OF) = 0 akkor
(IP) = (IP) + disp (előjelkiterjesztéssel 16 bitre)

ahol "disp" az utasítás operandusaként megadott címkének az utasítás végétől való távolsága.

Állított flag-ek: -

Példa: JNO közeli-címke

JNP *címke* vagy **JPO** *címke*

Ugrás a címkére, ha a parity flag értéke nulla. A vezérlést csak 128 byte-nál kisebb

távolságra lehet átadni. Előjeles ugrás.

A processzor lépése:

ha (PF) = 0 akkor
(IP) = (IP) + disp (előjelkiterjesztéssel 16 bitre)

ahol "disp" az utasítás operandusaként megadott címkének az utasítás végétől való távolsága.

Állított flag-ek:

Példa: JNP közeli-címke
JPO közeli-címke

JNS *címke*

Ugrás a címkére, ha az előjel flag értéke nulla. A vezérlést csak 128 byte-nál kisebb távolságra lévő címkére lehet átadni. Előjeles ugrás.

A processzor lépése:

ha (SF) = 0 akkor
(IP) = (IP) + disp (előjelkiterjesztéssel 16 bitre)

ahol "disp" az utasítás operandusaként megadott címkének az utasítás végétől való távolsága.

Állított flag-ek: -

Példa: JNS közeli-címke

JO *címke*

Ugrás a címkére, ha az overflow flag értéke 1. A vezérlést 128 byte-nál kisebb távoságra lévő címkére lehet átadni. Előjeles ugrás.

A processzor lépése:

ha (OF) = 1 akkor
(IP) = (IP) + disp (előjelkiterjesztéssel 16 bitre)

ahol "disp" az utasítás operandusaként megadott címkének az utasítás végétől való távolsága.

Állított flag-ek: -

Példa: JO közeli-címke

JP *címke* vagy **JPE** *címke*

Ugrás a címkére, ha a parity flag értéke 1. A vezérlést csak 128 byte-nál kisebb távolságra lévő címkére lehet átadni. Előjeles ugrás.

A processzor lépése:

ha (PF) = 1 akkor
(IP) = (IP) + disp (előjelkiterjesztéssel 16 bitre)

ahol "disp" az utasítás operandusaként megadott címkének az utasítás végétől való távolsága.

Állított flag-ek: -

Példa: JP közeli-címke
JPE közeli-címke

JS *címke*

Ugrás a címkére,ha az előjel flag értéke 1. A vezérlést csak a 128 byte-nál kisebb távolságra lévő címkére lehet átadni. Előjeles ugrás.

A processzor lépése:

ha (SF) = 1 akkor
(IP) = (IP) + disp (előjelkiterjesztéssel 16 bitre)

ahol "disp" az utasítás operandusaként megadott címkének az utasítás végétől való távolsága.

Állított flag-ek: -

Példa: JS közeli-címke

LAHF

A flag-ek regisztere alacsonyabb helyiértékű 8 bitjének az AH regiszterbe való töltése. Az AH regiszter 7., 6., 4., 2. és 0.bitjébe kerülnek az előjel, zéró, auxiliary carry, paritás és carry flagek.

A processzor lépése:

(AH) = (SF):(ZF):X:(AF):X:(PF):X:(CF)

X azt jelenti, hogy AH megfelelő helyiértékű bitje változatlan marad.

Állított flag-ek: -

LDS *reg,ált*

A második operandussal meghatározott memóriaszó tartalom az első operandussal megadott regiszterbe, az azt követő memóriaszó tartalma a DS regiszterbe töltődik.

A processzor lépései:

(reg) = ((ált))
(DS) = ((ált) + 2)

Állított flag-ek: -

Példa: LDS BX,tábla[SI]

LEA *reg, ált*

A második operandus által megadott relatív címet az első operandussal meghatározott regiszterbe tölti.

A processzor lépése:

(reg) = Effektív cím

Állított flag-ek: -

Példa: LEA DX,változó

LES *reg,ált*

Az utasítás a megcímzett memóriabeli szót az adott regiszterbe, a következő szót az ES regiszterbe tölti.

A processzor lépései:

(reg) = ((ált))
(ES) = ((ált) + 2))

Állított flag-ek: -

Példa: LES BX,tábla[SI]

LOCK

A busz zárolása a következő utasítás végrehajtásának idejére. Célja, hogy az adott utasítás végrehajtása alatt a memóriához más processzor ne férhessen hozzá.

Állított flag-ek: -

```
Példa: MOV AL,1                ;AL1-re állítása (azt jelenti, hogy zárolt)
       LOCK XCHG SEMO,AL       ;tesztelés és zárolás
       TEST AL,AL              ;a flag-ek beállítása AL szerint
       JNZ ell                 ;ismétlés, ha zárolt
       .
       .
       .
       MOV SEMO,0              ;a zárolás feloldása
```

LODS *forrás* vagy **LODSB** vagy **LODSW**

A DS:SI regiszterpárral megcímzett byte vagy szó tartalmát betölti az AL vagy AX regiszterbe. SI tartalma a direction flag állásától függően automatikusan módosul. (Lásd még a CLD, STD utasításokat.)

A processzor lépései:

(DEST) = (SRC)

ha (DF) = 0 akkor (SI) = (SI) + DELTA
ha (DF) = 1 akkor (SI) = (SI)-DELTA

Szavas művelet esetében:
SRC = (SI) + 1:(SI),DEST = AX,DELTA = 2

Byte-os művelet esetében:
SRC = (SI),DEST = AL,DELTA = 1

Állított flag-ek: -

```
Példa: CLD        ;törli az irányt jelző flag-et, így SI inkrementálódni fog
       MOV        ;SI,OFFSET byte-string
       LODS       ;byte-string ;SI = SI + 1
        .
        .
        .
       STD        ;DF-et úgy állítja be, hogy SI dekrementálódni fog
       MOV        ;SI OFFSET szó-string
       LODS       ;szó-string ;SI = SI-2
```

LOOP *címke*

Ciklusszervező utasítás. Dekrementálja a CX regiszter tartalmát, és ha CX tartalma nem nulla, vezérlésátadást hajt végre a címkére. A vezérlést csak 128 byte-nál kisebb távolságra lévő címkére lehet átadni. Vigyázat, CX értékének vizsgálata előtt történik CX dekrementálása! Lásd még JCXZ utasítást.

A processzor lépései:

(CX) = (CX)-1
ha (CX) < > 0 akkor
(IP) = (IP) + disp (előjelkiterjesztéssel 16 bitre)

ahol a "disp" az utasítás operandusaként megadott címkének az utasítás végétől való távolsága.

Állított flag-ek: -

Példa: egy nem-nulla tömb 16 bites ellenőrző összegének kiszámítása.

```
       MOV CX,LENGTH tömb
       MOV AX,0
       MOV SI,AX
       ;
next:  ADD AX,tömb[SI]
       ADD SI,TYPE tömb
       LOOP next
       MOV cks,AX
```

LOOPZ *címke* vagy **LOOPE** *címke*

Ciklusszervező utasítás. Dekrementálja a CX tartalmát; ha CX nem nulla és a zero flag

értéke 1, ugrást hajt végre a 128 byte-nál kisebb távolságra lévő címkére.

A processzor lépései:

(CX) = (CX)-1
ha (ZF) = 1 & (CX) < >0 akkor
(IP) = (IP) + disp (előjelkiterjesztéssel 16 bitre)

ahol a "disp" az utasítás operandusaként megadott címkének az utasítás végétől való távolsága.

Állított flag-ek: -

Példa: byte-os tömb első, nullától különböző elemének megkeresése.

```
        MOV CX,LENGTH tömb
        MOV SI,-1
next:   INC SI
        CMP tömb [SI],0
        LOOPE next
        JNE talált
        . teendők, ha a tömb minden
        . eleme nulla

talált: ;SI tartalmazza az első, nullától különböző elem indexét
```

LOOPNZ *címke* vagy **LOOPNE** *címke*

Ciklusszervező utasítás. Dekrementálja a CX tartalmát; ha CX nem nulla és a zero flag értéke nulla, ugrást hajt végre a 128 byte-nál kisebb távolságra lévő címkére.

A processzor lépései:

(CX) = (CX)-1

ha (ZF) = 0 & (CX) < >0 akkor
(IP) = (IP) + disp (előjelkiterjesztéssel 16 bitre)

ahol a "disp" az utasítás operandusaként megadott címkének az utasítás végétől való távolsága.

Állított flag-ek: -

Példa: 2 n byte hosszúságú vektor összegének kiszámítása. Az összegzést abbahagyjuk, ha mindkét összeadandó elem 0.

Ekkor a nullától különböző összegvektor hossza (SI)-1.

```
        MOV AX,0
        MOV SI,-1
        MOV CX,n
nem nulla: INC SI
        MOV AL,vektor1[SI]
        ADD AL,vektor2[SI]
```

```
MOV ossz[SI],AL
LOOPNZ nem-nulla
```

MOV *akku,mem*

8-vagy 16-bites adat betöltése a memóriából az akkumulátorba.

A processzor lépése:

(akku) = (mem)

Állított flag-ek: -

Példa: MOV AL,mem-byte
MOV AX,mem-szó

MOV *mem,akku*

8-vagy 16-bites adat töltése az akkumulátorból a memóriába.

A processzor lépése:

(mem) = (akku)

Állított flag-ek: -

Példa: MOV mem-byte,AL
MOV mem-szó,AX

MOV *ált,konst*

8-vagy 16- bites konstans vagy literál betöltése regiszterbe vagy memóriába.

A processzor lépése:

(ált) = konst

Állított flag-ek: -

Példa: MOV mem-szó,1234

MOV *ált,reg*

Regiszter tartalmának betöltése regiszterbe vagy memóriába.

A processzor lépése:

(ált) = (reg)

Állított flag-ek: -

Példa: MOV mem-szó,BX

MOV *ált,szreg*

Szegmensregiszter tartalmának betöltése regiszterbe vagy memóriába.

A processzor lépése:

(ált) = (szreg)

Állított flag-ek: -

Példa: MOV vektor[BX][SI],SS

MOV *reg,konst*

Konstans vagy literál betöltése regiszterbe.

A processzor lépése:

(reg) = (konst)

Állított flag-ek: -

Példa: MOV CX,100

MOV *reg,ált*

Memória vagy regiszter tartalmának betöltése regiszterbe.

A processzor lépése:

(reg) = (ált)

Állított flag-ek: -

Példa: MOV AX,mem-szó

MOV *szreg,ált*

Memória vagy regiszter tartalmának töltése szegmensregiszterbe. (A szegmensregiszter nem lehet CS.)

A processzor lépése:

(szreg) = (ált)

Állított flag-ek: -

Példa: MOV ES,SS:új-szó[DI]

MOVS *fogadó,küldő* vagy **MOVSB** vagy **MOVSW**

Karaktersorozat másolása a küldő mezőből a fogadó mezőbe byte-onként vagy szavanként. A küldő cím DS:SI, a fogadó ES:DI.

A processzor lépései:

(DEST) = (SCR)
ha (DF) = 0 akkor
(SI) = (SI) + DELTA
(DI) = (DI) + DELTA egyébként

(SI) = (SI)-DELTA
(DI) = (DI)-DELTA

Byte-os művelet esetén:
SRC = (SI), DEST:(DI), DELTA = 1

Szavas művelet esetén:
SRC = (SI) + 1:(SI), DEST = (DI) + 1:(DI), DELTA = 2

Állított flag-ek: -

Példa: MOV SI,OFFSET küldő
MOV DI,OFFSET fogadó
MOV CX,LENGTH küldő
REP MOVS fogadó,küldő
vagy
REP MOVS ES:BYTE PTR[DI],DS:[SI]
vagy
REP MOVSB

MOVSB

A DS:SI regiszterekkel megcímzett byte-ot átmásolja az ES:DI által meghatározott címre. Ha a REP prefixet is használjuk, akkor az ismétlési tényezőt CX tartalmazza.

MOVSW

A DS:SI által megcímzett szót másolja az ES:DI által meghatározott címre. Ha a REP prefixet is használjuk, akkor az ismétlési tényezőt CX tartalmazza.

MUL *ált*

Előjel nélküli bináris számok szorzása. Szavas szorzáskor az egyik tényezőt az AX regiszter tartalmazza, a másik tényező az utasítás operandusa. A szorzat a DX, AX regiszterpárba kerül. Byte-os szorzáskor az AL regiszter tartalmát szorozzuk az utasítás operandusával. A szorzat az AX regiszterbe kerül.

A processzor lépései:

(DEST) = (LSRC)*(RSRC)
ha (EXT) = 0 akkor
(CF) = 0 egyébként
(CF) = 1 , (OF) = (CF)

Byte-os műveletek esetén:
LSRC = AL, RSRC = ált, DEST = AX, EXT = AH

Szavas művelet esetén:
LSRC = AX, RSRC = ált, DEST = DX:AX, EXT = DX
Állított flag-ek: CF,OF
Példa: MOV AX,lsrc-szó
MUL rsrc-szó ; az eredmény a DX:AX regiszterpárba kerül.

NEG *ált*

Az operandus kettes komplemensének előállítása. Az eredeti érték negáltja + 1.

A processzor lépései:

(ált) = SRC-(ált)
(ált) = (ált) + 1

Byte-os művelet esetén SRC = 0FFH

Szavas művelet esetén SRC = 0FFFFH

Állított flag-ek: AF, CF, OF, PF, SF, ZF

Példa: NEG mem-byte
NEG mem-szó

NMI

Megszakítás következik be, ha a külső megszakítás nem maszkolható interrupt (NMI). Csak a 8088 mikroprocesszor utasításkészlete tartalmazza ezt az utasítást.

NOP

Inkrementálja a PC regiszter tartalmát, egyébként nincs hatása. Kódja megegyezik a MOV AX,AX utasítással.

A processzor lépése:

(IP) = (IP) + 1

NOT *ált*

Az operandus minden bitjét az ellentettjére állítja.

A processzor lépése:

(ált) = SRC-(ált)

Byte-os művelet esetén SRC = 0FFH

Szavas művelet esetén SRC = 0FFFFH

Állított flag-ek: -

Példa: NOT mem-byte
NOT mem-szó

OR *akku,konst*

Logikai "vagy" művelet az akkumulátor tartalma és a konstans vagy literál között. Az eredmény az akkumulátorba kerül.

A processzor lépései:

(akku) = (akku) IOR konst
(CF) = 0
(OF) = 0

Állított flag-ek: CF, OF, PF, SF, ZF

Példa: OR AX,123AH
OR AL,10110011B

OR *ált,konst*

Logikai "vagy" művelet memória vagy regiszter és konstans vagy literál között. Az eredmény az elsőként megadott operandusba kerül.

A processzor lépései:

(ált) = (ált) IOR konst
(CF) = 0
(OF) = 0

Állított flag-ek: CF, OF, PF, SF, ZF

Példa: OR mem-byte,2FH
OR mem-szó,98B6H

OR *ált,reg*

Logikai "vagy" művelet memória vagy regiszter és regiszter között. Az eredmény az elsőként megadott operandusba kerül.

A processzor lépései:

(ált) = (ált) IOR (reg)
(CF) = 0
(OF) = 0

Állított flag-ek: CF, OF, PF, SF, ZF

Példa: OR mem-byte,CL
OR mem-szó,DX

OR *reg,ált*

Logikai "vagy" művelet regiszter és regiszter vagy memória között. Az eredmény az elsőként megadott regiszterbe kerül.

A processzor lépései:

(reg) = (reg) IOR (ált)
(CF) = 0
(OF) = 0

Állított flag-ek:CF, OF, PF, SF, ZF

Példa: OR CH,mem-byte
OR AX,mem-szó

OUT DX,*akku*

Az akkumulátor tartalmát kiviszi a DX regiszter tartalma által megcímzett I/O portra.

A processzor lépése:

((DX)port) = (akku)

Állított flag-ek: -

Példa: OUT DX,AX

OUT *konst,akku*

Az akkumulátor tartalmát kiviszi az 1 byte-os konstans vagy literál által megcímzett I/O portra.

A processzor lépése:

(konst port) = (akku)

Állított flag-ek: -

Példa: OUT 44,AL

POP *ált*

A verem két felső byte-ját az utasításban megadott egyszavas operandusba tölti.

A processzor lépései:

(ált) = ((SP) + 1:(SP))
(SP) = (SP) + 2

Állított flag-ek: -

Példa: POP mem-szó

POP *reg*

A verem két felső byte-ját az utasításban megadott egyszavas regiszterbe tölti.

A processzor lépései:

(reg) = ((SP) + 1:(SP))
(SP) = (SP) + 2

Állított flag-ek: -

Példa: POP DX

POP *szreg*

A verem 2 felső byte-ját az utasításban megadott szegmensregiszterbe tölti. (A szegmensregiszter nem lehet CS.)

A processzor lépései:

(szreg) = ((SP) + 1:(SP))
(SP) = (SP) + 2

Állított flag-ek: -

Példa: POP SS

POPF

A verem 2 felső byte-ját a flag-ek regiszterébe tölti.

A processzor lépései:

flag-ek = ((SP) + 1:(SP))
(SP) = (SP) + 2

Állított flag-ek: minden flag

PUSH *ált*

Az utasításban megadott két byte-os operandus tartalmát a verem tetejére teszi.

A processzor lépései:

(SP) = (SP)-2
((SP) + 1:(SP)) = (ált)

Állított flag-ek: -

Példa: PUSH mem-szó

PUSH *reg*

Az utasításban megadott 16 bites regiszter tartalmát a verem tetejére teszi.

A processzor lépései:

(SP) = (SP)-2
((SP) + 1:(SP)) = (reg)

Állított flag-ek: -

Példa: PUSH CX

PUSH *szreg*

Az utasításban megadott szegmensregiszter tartalmát a verem tetejére teszi.

A processzor lépései:

(SP) = (SP)-2
((SP) + 1:(SP)) = (szreg)

Állított flag-ek: -

Példa: PUSH ES

PUSHF

A flag-ek 16 bites tartalmát a verem tetejére teszi.

A processzor lépései:

(SP) = (SP)-2
((SP) + 1:)SP)) = flag-ek

Állított flag-ek: -

RCL *ált*,**1**

Az operandus tartalmának ciklikus balra léptetése 1-gyel a carry flag-en keresztül.

A processzor lépései:

(tmpfc) = (CF)
(CF) = (ált) legmagasabb helyiértékű bitje
(ált) = (ált)*2 + (tmpcf)
ha (ált) legmagasabb helyiértékű bitje = (CF) akkor
(OF) = 0

Állított flag-ek: CF,OF

Példa: RCL mem-byte,1

RCL *ált*,**CL**

Az operandus tartalmának ciklikus balra léptetése a carry flag-en keresztül annyival, amennyi a CL regiszter tartalma.

A processzor lépései:

(temp) = (CL)
amíg (temp) < > 0
(tmpcf) = (CF)
(CF) = (ált) legmagasabb helyiértékű bitje
(ált) = (ált)*2 + (tmpcf)
(temp) = (temp)-1, majd
ha (CL) = 1 akkor
ha (ált) legmagasabb helyiértékű bitje < > (CF) akkor

(OF) = 1, egyébként (OF) = 0
egyébként (OF) definiálatlan.

Állított flag-ek: CF,OF

Példa: RCL változó[BX][DI],CL

RCR *ált*,**1**

Az operandus tartalmának ciklikus jobbra léptetése 1-gyel a carry flag-en keresztül.

A processzor lépései:

(tmpcf) = (CF)
(CF) = (ált) legalacsonyabb helyiértékű bitje
(ált) = (ált)/2*
(ált) legmagasabb helyiértékű bitje = (tmpfc)
ha (ált) legmagasabb helyiértékű bitje < >
(ált) köv. legmagasabb helyiértékű bitjével
akkor (OF) = 1 egyébként (OF) = 0

Állított flag-ek: CF, OF

Példa: RCR mem-byte,1

RCR *ált*,**CL**

Az operandus tartalmának ciklikus jobbra léptetése a carry flag-en keresztül annyival, amennyi a CL regiszter tartalma.

A processzor lépései:

(temp) = (CL)
amíg (temp) = 0
(tmpcf) = (CF)
(CF) = (ált) legalacsonyabb helyiértékű bitje
(ált) = (ált)/2*
(ált) legmagasabb helyiértékű bitje = (tmpcf)
(temp) = (temp)-1, majd
ha (CL) = 1 akkor
ha (ált) legmagasabb helyiértékű bitje =
(ált) köv. legmagasabb helyiértékű bitjével
akkor (OF) = 1 egyébként (OF) = 0
egyébként (OF) definiálatlan

REP

A prefixet követő karaktersorozatkezelő utasítás ismétlése, amíg a CX regiszter tartalma nem nulla.

A processzor lépései:

amíg (CX)< >0
függőben lévő megszakítás feldolgozása (ha létezik)
karakterkezelő utasítás végrehajtása a köv. byte-on
(CX) = (CX)-1

Állított flag-ek: ld. karakterkezelő utasítás

Példa: REP MOVS fogadó,küldő

REPE vagy **REPZ**

A prefixet követő karaktersorozatkezelő utasítás ismétlése, ameddig CX regiszter tartalma nem nulla, vagy a zero flag értéke nem egy.

A processzor lépései:

amíg (CX)< >0
függőben lévő megszakítás feldolgozása (ha létezik)
karakterkezelő utasítás végrehajtása a köv. byte-on
(CX) = (CX)-1
ha az utasítás CMPS vagy SCAS és (ZF) = 1
akkor kilépés a ciklusból.

Állított flag-ek: ld. karakterkezelő utasítás.

REPN vagy **REPNZ**

A prefixet követő karaktersorozatkezelő utasítás ismétlése, ameddig CX regiszter tartalma nem nulla, vagy zero flag értéke nem nulla.

A processzor lépései:

amíg (CX)< >0
függőben lévő megszakítás feldolgozása (ha létezik)
karakterkezelő utasítás végrehajtása a
köv. byte-on vagy szón
(CX) = (CX)-1
ha az utasítás CMPS vagy SCAS és (ZF) = 0
akkor kilépés a ciklusból.

Állított flag-ek: ld. karakterkezelő utasítás.

RET

Szubrutinból való visszatérés.

A processzor lépései:

(IP) = ((SP) + 1:(SP))
(SP) = (SP) + 2
ha külső szegmensből való visszatérés, akkor

(CS) = ((SP) + 1:(SP))
(SP) = (SP) + 2

Állított flag-ek: -

Példa: RET

RET *konst*

Szubrutinból való visszatérés. A verem értékhez a konstans értéke is hozzáadódik, így alakul ki a végleges cím.

A processzor lépései:

(IP) = ((SP) + 1:(SP))
(SP) = (SP) + 2
ha külső szegmensből való visszatérés, akkor
(CS) = ((SP) + 1:(SP))
(SP) = (SP) + 2
ha az utasításnak van operandusa, akkor
(SP) = (SP) + konst

Állított flag-ek: -

Példa: RET 12

ROL *ált*, **1**

Az operandus tartalmának ciklikus balra léptetése 1-gyel.

A processzor lépései:

(CF) = (ált) legmagasabb helyiértékű bitje
(ált) = (ált)*2 + (CF)
ha (ált) legmagasabb helyiértékű bitje < > (CF) akkor
(OF) = 1
egyébként (OF) = 0

Állított flag-ek: CF, OF

Példa: ROL mem-byte,1

ROL *ált*,**CL**

Az operandus tartalmának ciklikus balra léptetése annyival, amennyi a CL regiszter tartalma.

A processzor lépései:

(temp) = (CL)
amíg (temp) < > 0
(CF) = (ált) legmagasabb helyiértékű bitje
(ált) = (ált)*2 + (CF)
(temp) = (temp)-1, majd

ha (CL) = 1 akkor
ha (ált) legmagasabb helyiértékű bitje < > (CF) akkor
(OF) = egyébként (OF) = 0
egyébként (OF) = meghatározatlan

Állított flag-ek: CF, OF

Példa: ROL szó,CL

ROR *ált*,1

Az operandus tartalmának ciklikus jobbra léptetése 1-gyel.

A processzor lépései:

(CF) = (ált) legalacsonyabb helyiértékű bitje
(ált) = (ált)/2
(ált) legmagasabb helyiértékű bitje = (CF)
ha (ált) legmagasabb helyiértékű bitje < >
(ált) köv. legmagasabb helyiértékű bitjével
akkor (OF) = 1 egyébként (OF) = 0

Állított flag-ek: CF, OF

Példa: ROR mem-szó,1

ROR *ált*,CL

Az operandus tartalmának ciklikus jobbra léptetése annyival, amennyi a CL regiszter tartalma.

A processzor lépései:

(temp) = (CL)
amíg (temp) < > 0
(CF) = (ált) legalacsonyabb helyiértékű bitje
(ált) = (ált)/2
(ált) legmagasabb helyiértékű bitje = (CF)
(temp) = (temp)-1, majd
ha (CL) = 1 akkor
ha (ált) legmagasabb helyiértékű bitje < >
(ált) köv. legmagasabb helyiértékű bitjével
akkor (OF) = 1 egyébként (OF) = 0
egyébként (OF) definiálatlan

Állított flag-ek: CF,OF

Példa: ROR mem-szó,CL

SAHF

Az AH regiszter tartalmának betöltése a flag-ek regiszterének alacsonyabb helyiértékű 8 bitjére. A LAHF utasítás fordítottja.

A processzor lépése:

(SF):(ZF):X:(AF):x:(PF):X:(CF) = (AH)

X azt jelenti, hogy a flag-ek regiszterének megfelelő helyiértékű bitje változatlan marad.

Állított flag-ek: AF, CF, PF, SF, ZF

SAL *ált*,**1** vagy **SHL** *ált*,**1**

Az operandus tartalmának artimetikai balra léptetése 1-gyel.

A processzor lépései:

(CF) = (ált) legmagasabb helyiértékű bitje
(ált) = (ált)*2
ha (ált) legmagasabb helyiértékű bitje < > (CF) akkor
(OF) = 1 egyébként (OF) = 0

Állított flag-ek: CF, OF PF, SF, ZF

Példa: SAL változó,1
SHL mem-byte,1

SAL *ált*,**CL** vagy **SHL** *ált*,**CL**

Az operandus tartalmának artimetikai balra léptetése annyival, amennyi a CL regiszter tartalma.

A processzor lépései:

(temp) = (CL)
amíg (temp) < > 0
(CF) = (ált) legmagasabb helyiértékű bitje
(ált) = (ált)*2
(temp) = (temp)-1, majd
ha (CL) = 1 akkor
ha (ált) legmagasabb helyiértékű bitje < > (CF) akkor
(OF) = 1, egyébként
egyébként (OF) definiálaltlan

Állított flag-ek: CF, OF, PF, SF, ZF

Példa: SAL AX,CL
SHL mem-szó,CL

SAR *ált*,**1**

Az operandus tartalmának artimetikai jobbra léptetése 1-gyel.

A processzor lépései:

(CF) = (ált) legalacsonyabb helyiértékű bitje
(ált) = (ált)/2 előjeles osztás, lefelé kerekítéssel

ha (ált) legmagasabb helyiértékű bitje< >
(ált) köv. legmagasabb helyiértékű bitjével
akkor (OF) = 1 egyébként (OF) = 0

Állított flag-ek: CF,OF

Példa: SAR mem-byte,1

SAR *ált*,**CL**

Az operandus tartalmának artimetikai jobbra léptetése annyival, amennyi a CL regiszter tartalma.

A processzor lépései:

(temp) = 1
amíg (temp)< >0
(CF) = (ált) legalacsonyabb helyiértékű bitje
(ált) = (ált)/2 (előjeles osztás, lefelé kerekítés)
(temp) = (temp)-1
ha (CL) = 1 akkor
ha (ált) legmagasabb helyiértékű bitje< >
(ált) köv. legmagasabb helyiértékű bitjével,
egyébként (OF) = 0

Állított flag-ek: CF, OF, PF, SF, ZF

Példa: SAR AX, CL

SBB *akku,konst*

Konstans vagy literál és a carry flag kivonása az akkumulátor tartalmából. Az eredmény az akkumulátorba kerül.

A processzor lépései:

ha (CF) = 1 akkor (akku) = (akku)-konst-1
ha (CF) = 0 akkor (akku) = (akku)-konst.

Állított flag-ek: AF, CF, OF, PF, SF, ZF

Példa: SBB AX,432

SBB *ált,konst*

Konstans vagy literál és a carry flag kivonása egy memóriacím vagy egy regiszter tartalmából. Az eredmény az adott című területre vagy a regiszterbe kerül.

A processzor lépései:

ha (CF) = 1 akkor (ált) = (ált)-konst-1
ha (CF) = 0 akkor (ált) = (ált)-konst

Állított flag-ek: AF, CF, OF, PF, SF, ZF

Példa: SBB mem-szó,123

SBB *ált,reg*

Regisztertartalom és a carry flag kivonása regiszter vagy adott című memóriaterület tartalmából. Az eredmény az elsőként felírt operandusba kerül.

A processzor lépései:

ha (CF) = 1 akkor (ált) = (ált)-(reg)-1
ha (CF) = 0 akkor (ált) = (ált)-(reg)

Állított flag-ek: AF, CF, OF, PF, SF, ZF

Példa: SBB mem-szó,AX

SBB *reg,ált*

Regisztertartalom vagy adott című terület és a carry flag tartalmának kivonása egy regiszter tartalmából. Az eredmény az elsőként felírt operandusba kerül.

A processzor lépései:

ha (CF) = 1 akkor (reg) = (reg)-(ált)-1
ha (CF) = 0 akkor (reg) = (reg)-(ált)

Állított flag-ek: AF, CF, OF, PF, SF, ZF

Példa: SBB AX,mem-szó

SCAS *string*

Az ES:DI regiszterek tartalmával címzett byte vagy szó összehasonlítása az AL vagy AX regiszter tartalmával. A művelet mindkét operandusa változatlan marad, csak a flag-eket állítja.

A processzor lépései:

(LSRC)-(RSRC)
ha (DF) = 0 akkor (DI) = (DI) + DELTA
egyébként (DI) = (DI)-DELTA

Byte-os művelet esetén:
LSRC = AL, RSRC = (DI),DELTA = 1

Szavas művelet esetén
LSRC = AX,RSRC = (DI) + 1:(DI),DELTA = 2

Állított flag-ek: AF, CF, OF, PF, SF, ZF

```
Példa: CLD  ;törli DF-et, így DI inkrementálódik majd
       MOV  DI OFFSET byte-string
       MOV  AL, 'M'
```

```
        SCAS byte-string
vagy
        SCAS ES:BYTE PTR[DI]
vagy
        SCASB
        STD  ;beállítja DF-et, így DI dekrementálódik majd
        MOV  DI OFFSET szó-string
        MOV  AX, 'MD'
        SCAS szó-string
```

SHL *ált*,**1** vagy **SAL** *ált*,**1**

Az operandus tartalmának logikai balra léptetése 1-gyel.

A processzor lépései:

```
        (CF) = (ált) legmagasabb helyiértékű bitje
        (ált) = (ált)*2
ha      (ált) legmagasabb helyiértékű bitje < > (CF)
        akkor (OF) = 1 egyébként (OF) = 0
```

Állított flag-ek: CF, OF, PF, SF, ZF

Példa: SAL változó,1
SHL mem-byte,1

SHL *ált*,**CL** vagy **SAL** *ált*,**CL**

Az operandus tartalmának logikai balra léptetése annyival, amennyi a CL regiszter tartalma.

A processzor lépései:

```
        (temp) = (CL)
amíg    (temp) < >0
        (CF) = (ált) legmagasabb helyiértékű bitje
        (ált) = (ált)*2
        (temp) = (temp)-1, majd
 ha     (CL) = 1 akkor
        ha      (ált) legmagasabb helyiértékű bitje< > (CF)
                akkor (OF) = 1 egyébként (OF) = 0
egyébként (OF) definiálatlan.
```

Állított flag-ek: CF, OF, PF, SF, ZF

Példa: SAL AX,CL
SHL mem-szó,CL

SHR *ált*,**1**

Az operandus tartalmának logikai jobbra léptetése 1-gyel.

A processzor lépései:

(CF) = (ált) legalacsonyabb helyiértékű bitje
(ált) = (ált)/2 (előjeles osztás, lefelé kerekítés)
ha (ált) legmagasabb helyiértékű bitje < > (ált) köv legmagasabb helyiértékű bitjével
akkor (OF) = 1
egyébként (OF) = 0

Állított flag-ek: CF, OF, PF, SF, ZF

Példa: SHR mem-byte,1

SHR *ált*,CL

Az operandus tartalmának logikai jobbra léptetése annyival, amennyi a CL regiszter tartalma.

A processzor lépései:

(temp) < > (CL)
amíg (temp) = 0
(CF) = (ált) legalacsonyabb helyiértékű bitje
(ált) = (ált)/2 (előjeles osztás, lefelé kerekítés)
(temp) = (temp)-1, majd
ha (CL) = 1 akkor
ha (ált) legmagasabb helyiértékű bitje =
(ált) köv. legmagasabb helyiértékű bitjével
akkor (OF) = 1 egyébként OF = 0
egyébként (OF) definiálatlan

Állított flag-ek: CF, OF, PF, SF, ZF

Példa: SHR mem-szó,CL

STC

Az átviteli (carry) flag 1-re állítása.

A processzor lépése:

(CF) = 1

Állított flag: CF

STD

A feldolgozás irányát jelző (direktion) flag 1-re állítása. Ekkor az SI és/vagy DI regisztereket automatikusan állító utasítások autodecrement módban dolgoznak (pl. LODS, STOS)

A processzor lépése:
(DF) = 1
Állított flag: DF

STI

A megszakítást engedélyező (interrupt) flag 1-re állítása.

A processzor lépése:

(IF) = 1

Állított flag: IF

STOS *string*, **STOSB, STOSW**

AL vagy az AX regiszter tartalmának tárolása az ES:DI regiszterek tartalmával megcímzett byte-on vagy szón. Az utasítás végrehajtása után DI értéke a direction flag állásától függően eggyel csökken vagy nő.

A processzor lépései:

(DEST) = (SCR)
ha (DF) = 0 akkor (DI) = (DI) + DELTA
egyébként (DI) = (DI)-DELTA

Byte-os művelet esetén:
SRC = AL,DEST = (DI),DELTA = 1

Szavas művelet esetén:
SRC = AX,DEST = (DI) + 1:(DI),DELTA = 2

Állított flag-ek: -

Példa: MOV DI,OFFSET byte-string
STOS BYTE-string vaagy STOSB
MOV DI,OFFSET szó-string
STOS szó-string vagy STOSW

SUB *akku,konst*

Konstans vagy literál kivonása az akkumulátor tartalmából. Az eredmény az akkumulátorba kerül.

A processzor lépése:

(akku) = (akku)-konst

Állított flag-ek: AF, CF, OF, PF, SF, ZF

Példa: SUB AL,3

SUB *ált,konst*

Konstans vagy literál kivonása egy adott című memóriaterület vagy regiszter tartalmából. Az eredmény az elsőként felírt operandusba kerül.

A processzor lépése:

(ált) = (ált)-konst

Állított flag-ek: AF, CF, OF, PF, SF, ZF

Példa: SUB mem-szó[DI][BX],1234

SUB *ált,reg*

Regisztertartalom kivonása egy adott című memóriaterület vagy regiszter tartalmából. Az eredmény az elsőként felírt operandusba kerül.

A processzor lépése:

(ált) = (ált)-(reg)

Állított flag-ek: AF, CF, OF, PF, SF, ZF

Példa: SUB mem-szó,CX

SUB *reg,ált*

Adott című memóriaterület vagy egy regiszter tartalmának kivonása egy regiszter tartalmából. Az eredmény az elsőként felírt operandusba kerül.

A processzor lépése:

(reg) = (reg)-(ált)

Állított flag-ek: AF, CF,OF, PF, SF, ZF

Példa: SUB BX,mem-szó

TEST *akku,konst*

Logikai "és" művelet konstans vagy literál és az akkumulátor tartalma között. A művelet operandusai nem változnak, a flag-eket állítja.

A processzor lépései:

(akku)&konst
(CF) = 0
(OF) = 0

Állított flag-ek: CF, OF, PF, SF, ZF

Példa: TEST AL,2

TEST *ált,konst*

Logikai "és" művelet konstans vagy literál és egy adott című memóriaterület vagy egy regiszter tartalma között. A művelet operandusai nem változnak, a flag-eket állítja.

A processzor lépései:

(ált)&konst

(CF) = 0
(OF) = 0

Állított flag-ek: CF, OF, PF, SF, ZF

Példa: TEST mem-szó 532

TEST *ált,reg*

Logikai "és" művelet egy regiszter tartalma és egy adott című memóriaterület vagy egy regiszter tartalma között. A művelet operandusai nem változnak, a flag-eket állítja.

A processzor lépései:

(ált)&(reg)
(CF) = 0
(OF) = 0

Állított flag-ek: CF, OF, PF, SF, ZF

Példa TEST mem-szó,SI

TEST *reg,ált*

Logikai "és" művelet egy adott című memóriaterület vagy egy regiszter tartalma és egy regisztertartalom között. A művelet operandusai nem változnak, a flag-eket állítja.

A processzor lépései:

(reg)&(ált)
(CF) = 0
(OF) = 0

Állított flag-ek: CF, OF, PF, SF, ZF

Példa: TEST DI,mem-szó

WAIT

Az utasítások végrehajtása addig szünetel, amíg a processzor "TEST" bemenet szintje nulla. Ilyenkor az egyik processzor várhat addig, amíg a másik bizonyos műveletet be nem fejez.

Állított flag-ek: -

XCHG *akku,reg*

Az akkumulátor és egy regiszter tartalmának megcserélése.

A processzor lépései:

o(temp) = (akku)
(akku) = (reg)
(reg) = (temp)

Állított flag-ek: -

Példa: XCHG AX,BX

XCHG *ált,reg*

Egy adott című memóriaterület vagy egy regiszter tartalmának megcserélése egy regiszter tartalmával.

A processzor lépései:

(temp) = (ált)
(ált) = (reg)
(reg) = (temp)

Állított flag-ek: -

Példa: XCHG mem-szó,CX

XCHG *reg,akku*

Egy regiszter tartalmának megcserélése az akkumulátor tartalmával.

A processzor lépései:

(temp) = (reg)
(reg) = (akku)
(akku) = (temp)

Állított flag-ek: -

Pelda: XCHG DX,AX

XCHG *reg,ált*

Egy regiszter tartalmának megcserélése egy adott című memóriaterület vagy egy regiszter tartalmával.

A processzor lépései:

(temp) = (reg)
(reg) = (ált)
(ált) = (temp)

Állított flag-ek: -

Példa: XCHG DX,mem-szó

XLAT *mem*

A DS:(BX) + (AL) regiszterek tartalma által megcímzett byte az AL regiszterbe kerül. Rendszerint akkor használjuk, ha táblázatból akarunk adatokat behozni AL-be. Ekkor DS:BX mutat a táblázat elejére, AL pedig indexként rámutat a táblázat adott elemére.

A processzor lépése:

(AL) = ((BX)) + (AL))

Állított flag-ek: -

Példa: MOV AL,5
MOV BX,OFFSET táblanév
XLAT táblanév

XOR *akku,konst*

"Kizáró vagy" művelet egy konstans vagy literál és az akkumulátor tartalma között. Az eredmény az akkumulátorba kerül.

A processzor lépései:

(akku) = (akku) XOR konst
(CF) = 0
(OF) = 0

Állított flag-ek: CF, OF, PF, SF, ZF

Példa: XOR AX,512

XOR *ált,konst*

"Kizáró vagy" művelet egy konstans vagy literál és egy adott című memóriaterület vagy egy regiszter tartalma között. Az eredmény az elsőként felírt operandusba kerül.

A processzor lépései:

(ált) = (ált) XOR konst
(CF) = 0
(OF) = 0

Állított flag-ek: CF, OF, PF, SF, ZF

Példa: XOR mem-byte,1DH

XOR *ált,reg*

"Kizáró vagy" művelet egy regiszter tartalma és egy adott című memóriaterület vagy egy regiszter tartalma között. Az eredmény az elsőként felírt operandusba kerül.

A processzor lépései:

(ált) = (ált) XOR (reg)
(CF) = 0
(OF) = 0

Állított flag-ek: CF, OF, PF, SF, ZF

Példa: XOR mem-szó,BX

XOR *reg,ált*

"Kizáró vagy" művelet regiszter vagy memóriaterület tartalma és egy regiszter tartalma között. Az eredmény az elsőként felírt operandusba kerül

A processzor lépései:

(reg) = (reg) XOR (ált)
(CF) = 0
(OF) = 0

Állított flag-ek: CF, OF, PF, SF, ZF

Példa: XOR DX, mem-szó

4.3 A 8087 utasításkészlete

A 8087 lebegőpontos koprocesszor a 8086/8088 mikroprocesszor utasításkészletét bővíti. A 8087 lehetővé teszi a lebegőpontos és nagy pontosságot igénylő számítások elvégzését.

4.3.1 A 8087-es numerikus adat-processzor felépítése

Adat formátumok

adattípus	**bitek**
egész(WI) kettes kompl.	0 - 15
egész(SI) kettes kompl.	0 - 31
egész(LI) kettes kompl.	0 - 63
binárisan kódolt dec.(PD)	0 - 79
valós(SR)	0 - 31
valós(LR)	0 - 63
valós(TR)	0 - 79

Az egyes adattípusok felépítése bitenként:

WI: 0 - 14. érték, 15. előjel.
SI: 0 - 30. érték, 31. előjel.
LI: 0 - 62. érték, 63. előjel.
PD: 0 - 71. érték, 72-78. nem használt, 79. előjel.
SR: 0 - 22. alap, 23-30. kitevő, 31. előjel.
LR: 0 - 51. alap, 52-62. kitevő, 63. előjel.
TR: 0 - 63. alap, 64-78. kitevő, 79. előjel.

Az SR és LR adattípus esetében az alap legnagyobb helyiértékű bitje nincs ábrázolva, mert értéke mindig 1.

A regiszterek felépítése

A 8087-es matematikai koprocesszor 8 db., egyenként 80 bites regisztert használ az adatok kezelésére, plusz minden regiszter mellett egy 2 bites TAG regisztert, mely a belső

regiszterek foglaltságát jelzi.

A regiszterek bit felosztása:

0 - 63. alap
64 - 78. kitevő
79. előjel bit

Egyéb regiszterek:

2 db 16 bites vezérlő regiszter
2 db 16 bites program számláló
2 db 16 bites adat mutató.

A vezérlő regiszterek felépítése:

Az egyik vezérlő regiszter input, a másik output üzemmódú.

Az **input** vezérlő regiszter felépítése:
0. bit: érvénytelen művelet
1. bit: denormalizált operandus
2. bit: nullával való osztás
3. bit: túlcsordulás
4. bit: pontosság
5. bit: nem használható
6. bit: megszakítás engedélyezés
7. - 8. bit: pontosság vezérlés
9. - 10. bit: kerekítés vezérlés
11. bit: ábrázolhatatlanul nagy szám kezelése
12. - 15. bit: nem használható

Az **output** (státusz) regiszter felépítése:
0. bit: érvénytelen művelet
1. bit: denormalizált operandus
2. bit: nullával való osztás
3. bit: túlcsordulás
4. bit: alulcsordulás
5. bit: pontosság
6. bit: nem használható
7. bit: megszakítás kérés
8. - 10. és a 14. bit: feltétel kódok
11 - 13. bit: verem (adat regiszterek) teteje mutató
15. bit: processzor foglaltságának jelzése

Utasítás készlet

Minden utasítás az ESC kóddal kezdődik, mivel innen "tudja" a koprocesszor, hogy a következő parancsok neki szólnak.

Az ESC kódja: 11011xxx, ahol xxx már az utasítás része.

A. Adatátviteli utasítások

FLD: értékek koprocesszor adatterületre töltése.
Általános forma:
(ESC mf 1)(mod 0 0 0 r/m)(LO byte)(HI byte)
Speciális forma:
Adat az LI regiszterbe:
(ESC 1 1 1)(mod 1 0 1 r/m)(LO byte)(HI byte)
Adat a TR regiszterbe:
(ESC 0 1 1)(mod 1 0 1 r/m)(LO byte)(HI byte)
Adat a PD regiszterbe:
(ESC 1 1 1)(mod 1 0 0 r/m)(LO byte)(HI byte)
Regiszter értéke a legfelső regiszterbe:
(ESC 0 0 1)(1 1 0 0 0 st(i))

FST: regiszterek memóriába töltése a megadott adattípus szerint (a legfelső regiszter értéke változatlan marad).
Általános forma:
(ESC mf 1)(mod 0 1 0 r/m)(LO byte)(HI byte)
Speciális forma:
A regiszter teteje másik regiszterbe:
(ESC 0 1 1)(1 1 0 1 0 st(i))

FSTP: ugyanaz, mint az FST, de a legfelső regiszter tartalma elvész.
Általános forma:
(ESC mf 1)(mod 0 1 1 r/m)(LO byte)(HI byte)
Speciális forma:
LI regiszter tartalmának memóriába töltése:
(ESC 1 1 1)(mod 1 1 1 r/m)(LO byte)(HI byte)
TR regiszter tartalmának memóriába töltése:
(ESC 0 1 1)(mod 1 1 1 r/m)(LO byte)(HI byte)
PD regiszter tartalmának memóriába töltése:
(ESC 1 1 1)(mod 1 1 0 r/m)(LO byte)(HI byte)
A regiszter teteje másik regiszterbe:
(ESC 0 0 1)(1 1 0 0 1 st(i))

FXCH: st(0) és st(i) tartalmának kicserélése:
(ESC 0 0)(1 1 0 0 1 ST(i))

B. Összehasonlító utasítások

FCOM: összehasonlítás (legfelső regiszter értéke változatlan marad).
Általános forma:
(ESC mf 0)(mod 0 1 0 r/m)(LO byte)(HI byte)
st(0) összehasonlítása st(i)-vel:
(ESC 0 0 0)(1 1 0 1 0 st(i))

FCOMP: összehasonlítás (a legfelső regiszter tartalma elvész).
Általános forma:
(ESC mf 0)(mod 0 1 1 r/m)(LO byte)(HI byte)
st(0) összehasonlítása st(i)-vel:
(ESC 0 0 0)(1 1 0 1 1 st(i))

FCOMPP: a legfelső két regiszter értékének összehasonlítása és elvesztése.
(ESC 1 1 0)(1 1 0 1 1 0 0 1)

FTST: a felső regiszter értékének összehasonlítása nullával.
(ESC 0 0 1)(1 1 1 0 0 1 0 0)

FXAM: az input vezérlő regiszter 0-3. bitjét állítja a felső regiszterben lévő adat típusa szerint.
(ESC 0 0 1)(1 1 1 0 0 1 0 1)

C. Aritmetikai utasítások

FADD: összeadás
Általános forma:
(ESC mf 0)(mod 0 0 0 r/m)(LO byte)(HI byte)
Speciális forma:
A legfelső regiszter összeadása egy másikkal:
(ESC d p 0)(1 1 0 0 0 ST(i))

FSUB: kivonás
Általános forma:
(ESC mf 0)(mod 1 0 R r/m)(LO byte)(HI byte)
Speciális forma:
regiszter tartalmának kivonása a legfelső regiszterből, vagy fordítva:
(ESC d p 0)(1 1 1 0 R ST(i))

FMUL: szorzás
Általános forma:
(ESC mf 0)(mod 0 0 1 r/m)(LO byte)(HI byte)
Speciális forma:
a legfelső regiszter szorzása egy másikkal:
(ESC d p 0)(1 1 0 0 1 ST(i))

FDIV: osztás
Általános forma:
(ESC mf 0)(mod 1 1 R r/m)(LO byte)(HI byte)
Speciális forma:
a legfelső regiszter osztása egy másikkal:
(ESC d p 0)(1 1 1 1 R ST(i))

D. Függvények

SQRT: a legfelső regiszter tartalmának négyzetgyöke.
(ESC 0 0 1)(1 1 1 1 1 1 0 1)

FSCALE: ST(0)*2|ST(1)
(ESC 0 0 1)(1 1 1 1 1 1 1 0)

FPREM: ST(0)/ST(1) maradéka
(ESC 0 0 1)(1 1 1 1 1 0 0 0)

FRNDINT: ST(0) kerekített egész értéke.
(ESC 0 0 1)(1 1 1 1 1 1 0 0)

FXTRACT: ST(0) alapja ST(0)-ban marad, a kitevő ST(1)-be kerül.
(ESC 0 0 1)(1 1 1 1 0 1 0 0)

FABS: ST(0) abszolút értékét adja.
(ESC 0 0 1)(1 1 1 0 0 0 0 1)

FCHS: előjel váltása.
(ESC 0 0 1)(1 1 1 0 0 0 0 0)

E. Trigonometrikus és logaritmus függvények

FPTAN: ST(0) parciális tangense.
(ESC 0 0 1)(1 1 1 1 0 0 1 0)

FPATAN: ST(0) parciális arcus tangense.
(ESC 0 0 1)(1 1 1 1 0 0 1 1)

F2XM1: 2|ST(0) mínusz 1
(ESC 0 0 1)(1 1 1 1 0 0 0 0)

FYL2X: ST(1)*log2(ST(0))
(ESC 0 0 1)(1 1 1 1 0 0 0 1)

FYL2XP1: ST(1)*log2(ST(0) + 1)
(ESC 0 0 1)(1 1 1 1 1 0 0 1)

F. Konstansok

FLDZ: ST(0) = 0
(ESC 0 0 1)(1 1 1 0 1 1 1 0)

FLD1: ST(0) = 10
(ESC 0 0 1)(1 1 1 0 1 0 0 0)

FLDPI: ST(0) = PI értékével.
(ESC 0 0 1)(1 1 1 0 1 0 1 0)

FLDL2T: ST(0) = log2(10)
(ESC 0 0 1)(1 1 1 0 1 0 0 1)

FLDL2E: ST(0) = log2(e)
(ESC 0 0 1)(1 1 1 0 1 0 1 0)

FLDLG2: ST(0) = log10(2)
(ESC 0 0 1)(1 1 1 0 1 1 0 0)

FLDLN2: ST(0) = loge(2)
(ESC 0 0 1)(1 1 1 0 1 1 0 1)

Vezérlő utasítások

FINIT: 8087 inicializálása
(ESC 0 1 1)(1 1 1 0 0 0 1 1)

FENI: megszakítás engedélyezés
(ESC 0 1 1)(1 1 1 0 0 0 0 0)

FDISI: megszakítás letiltása
(ESC 0 1 1)(1 1 1 0 0 0 0 1)

FLDCW: 8087 vezérlő regiszter betöltése a memóriából
(ESC 0 0 1)(mod 1 0 1 r/m)(LO byte)(HI byte)

FSTCW: 8087 vezérlő regiszter a memóriába
(ESC 0 0 1)(mod 1 1 1 r/m)(LO byte)(HI byte)

FSTSW: 8087 státusz regiszter a memóriába
(ESC 1 0 1)(mod 1 1 1 r/m)(LO byte)(HI byte)

FCLEX: státusz regiszter törlése
(ESC 0 1 1)(1 1 1 0 0 0 1 0)

FSTENV: a 8087 összes regiszterének elmentése a memóriába, kivéve az adat regisztereket
(ESC 0 0 1)(mod 1 1 0 r/m)(LO byte)(HI byte)

FLDNEV: a 8087 regisztereinek betöltése a memóriából (kivéve az adat regisztereket)
(ESC 0 0 1)(mod 1 0 0 r/m)(LO byte)(HI byte)

FSAVE: a 8087 állapotának elmentése
(ESC 1 0 1)(mod 1 1 0 r/m)(LO byte)(HI byte)

FRSTOR: a 8087 állapotának visszatöltése a memóriából
(ESC 1 0 1)(mod 1 0 0 r/m)(LO byte)(HI byte)

FINCSTP: adatregiszter mutató értékének növelése 1-gyel
(ESC 0 0 1)(1 1 1 1 0 1 1 1)

FDECSTP: adatregiszter mutató értékének növelése 1-gyel
(ESC 0 0 1)(1 1 1 1 0 1 1 0)

FFREE: adatregiszter tartalmának törlése
(ESC 1 0 1)(1 1 0 0 0 ST(i))

FNOP: üres utasítás
(ESC 0 0 1)(1 1 0 1 0 0 0 0)

FWAIT: várakozó utasítás
(1 0 0 1 1 0 1 1)

Jelmagyarázat

mod értékei:
00 - nincsenek a címzésben LO HI byte-k
01 - előjeles LO byte van
10 - LO és HI byte van
11 - r/m ST(i)-nek felel meg

r/m értékei:
000 - EA = (BX) + (SI) + byte-k
001 - EA = (BX) + (DI) + byte-k
010 - EA = (BP) + (SI) + byte-k
011 - EA = (BP) + (DI) + byte-k
100 - EA = (SI) + byte-k
101 - EA = (DI) + byte-k
110 - EA = (BP) + byte-k
111 - EA = (BX) + byte-k

mf értékei:
00 - 32 bites valós szám
01 - 32 bites egész szám
10 - 64 bites valós szám
11 - 16 bites egész szám
ST(0) - az aktuális adatregiszter
ST(i) - az i. adatregiszter
d - 0 esetén célregiszter az ST(0),
1 esetén az ST(i).
p - 0 esetén a regiszter nem változik,
1 esetén tartalma elvész.
r - 1 esetén a művelethez tartozó tagok felcserélődnek.

4.4 A 80186 és a 80286 utasításkészlete

A 80186-os és a 80286-os mikroprocesszorok utasításkészlete a 8086/8088 mikroprocesszor teljes utasításkészletét, valamint a következő utasításokat tartalmazza:

BOUND *reg,mem*
Azt kérdezi le, hogy egy érték eleme-e valamely intervallumnak.

ENTER *konst 16,konst 8*
Eljárásba való belépés.

INS *mem*,**DX**
Karaktersorozat beolvasása a DX portról.

INSB *mem*,**DX**
Byte-os karaktersorozat beolvasása a DX portról.

INSW *mem*,**DX**
Szavas karaktersorozat beolvasása a DX portról.

LEAVE
Eljárásból való kilépés.

OUTS DX,*mem*
Byte, szó vagy karaktersorozat kiírása a DX portra.

OUTSB DX,*mem*
Byte-os karaktersorozat kiírása a DX portra.

OUTSW DX,*mem*
Szavas karaktersorozat kiírása a DX portra.

PUSHA
Az összes regiszter elhelyezése a veremben.

POPA
Az összes regiszter kivétele a veremből.
A 286-os mikroprocesszor az előzőekben leírt utsításokon kívül privilégizált utasításokat is tartalmaz. Ezek a következők:

ARPL *mem,reg*
A kívánt privilégium-szint beállítása.

CLTS
A taszk-kapcsolt flag törlése.

LAR *reg,mem*
A hozzáférési jogok betöltése.

LGDT *mem*
A 8-byte-os globális leíró tábla betöltése.

LIDT *mem*
A megszakítás-leíró tábla betöltése.

LLDT *mem*
A lokális leíró tábla betöltése.

LMSW *mem*
A gépi állapotjelző szó betöltése.

LSL *reg,mem*
A szegmenshatár betöltése.

LTR *mem*
A taszk-regiszter betöltése.

SGDT *mem*
A 8-byte-os globális leíró tábla tárolása.

SIDT *mem*
A 8-byte-os megszakítás-leíró tábla tárolása.

SLDT *mem*
A lokális leíró tábla tárolása.

SMSW *mem*
A gépi állapotjelző szó tárolása.

STR *mem*
A taszk-regiszter tárolása.

VERR *mem*
Az olvasásra való hozzáférés ellenőrzése.

VERW *mem*
Az írásra való hozzáférés ellenőrzése.

4.5 A 80287 utasításkészlete

A 287 mikroprocesszor utasításkészlete a 8087 teljes utasításkészletét, valamint a következő utasításokat tartalmazza:

FSETPM
Védett mód beállítása.

FSTSW AX
Az állapotjelző szó tárolása AX-ben WAIT-tel.

FNSTSW AX
Az állapotjelző szó tárolása AX-ben WAIT nélkül.

5. Programozás macro-assembly nyelven

5.1 A macro-assembly programok felépítése

A macro-assembly programok modulokból, a modulok szegmensekből, a szegmensek assembly utasításokból, direktívákból és makró-direktívákból állnak. Egy **modul** azon szegmensek összessége, amelyek a fordítás során egy tárgykód (.OBJ) file-ba kerülnek. A **szegmens** a SEGMENT és az ENDS direktívák közti programrészlet. Az assembly utasítások, a direktívák és a makró-direktívák használatának szabályait ismerteti a következő fejezet.

A program szerkezetét, a modulok egymáshoz viszonyított és abszolút elhelyezkedését a memóriában a programozónak kell meghatározni a direktívák segítségével. Ezek a direktívák a következők:

SEGMENT	szegmens definiálása
ENDS	szegmens vége
END	forrás-file vége
GROUP	szegmens-csoportok
ASSUME	szegmens-regiszterek
ORG	szegmens kezdőcíme
EVEN	szegmens szóhatárra-igazítása
PROC	eljárás definiálása
ENDP	eljárás vége

A direktíváknak lehetnek paraméterei, ezeket a továbbiakban attribútumoknak nevezzük.

5.1.1 A macro-assembly utasítások általános alakja

A makró-assembly utasítások hatására a fordító valamilyen tevékenységet hajt végre, pl. gépi kódú utasításokat vagy adat-byte-okat állít elő. Az utasítások általános alakja:

[*címke*:][*mnemonik*[*operandus*,,,]][;*megjegyzés*]

Az utasítások nem lehetnek többsorosak és hosszuk nem haladhatja meg a 128 karaktert. Tetszőleges pozíción kezdődhetnek a képernyőn. Az utasítás végét a képernyőn és a lemezen való tároláskor a soremelés karakter jelzi. Ez a forrás-file utolsó utasítására is vonatkozik!

Példa:

alprog: MOV CX,LENGTH vektor ; ismeretlen tenyezo

5.1.2 Az utasításkészletet beállító direktívák

Az utasításkészletet beállító direktívák a következők:

.8086
.8087

.186
.286c
.286p
.287

Ezek a direktívák engedélyezik az adott mikroprocesszor utasításkészletének használatát. Az utasításkészlet-direktívának a program elején kell állnia, ha használjuk.

A fordító alapértelmezésben a 8086 és a 8087 utasításkészletét fordítja, ezért a .8086 és .8087 direktívákat nem kötelező megadni.A .8086 direktíva a 8086 mikroprocesszor utasításkészletének használatát engedélyezi és letiltja a 80186 és 80286 utasításait. A .8087 direktíva a 8087 lebegőpontos társprocesszor utasításainak használatát engedélyezi és letiltja a 80287 utasításait.

A .186 direktíva engedélyezi a 80186 mikroprocesszor utasításainak fordítását.

A .286c direktíva a 80286 mikroprocesszor nem-privilégizált utasításainak használatát engedélyezi. Ezek az utasítások megegyeznek a 80186 utasításaival. A .286p direktíva kiadása után a 80286 mikroprocesszor privilégizált utasításai is használhatók.

A .287 direktíva hatására a 80287 lebegőpontos koprocesszor utasításait fordítja a makró-assembler.

A mikroprocesszor-család assembly utasításainak leírását a negyedik fejezet tartalmazza.

Amikor a .8087 és .287 direktívát használjuk, a fordító indításánál (ld. VII. fejezet) meg kell adni a /R vagy a /E opciót. A /R opció hatására lebegőpontos utasítás esetén a fordító az adott utasítás kódját generálja, /E opció hatására viszont software interrupt kódot állít elő a megfelelő lebegőpontos emulátorrutin hívásához.

5.1.3 A forrás-modulok és az END direktíva

Minden assembly nyelvű program egy vagy több forrás-modulból áll. *A modulok (forrás-file-ok) egyszerű szöveg-file-ok, ezekből a makró-assembler az utasítások lefordításával tárgymodulokat állít elő.* **Minden forrás-file azonos szerkezetű: nulla vagy több programszegmensből és a file végét jelző END direktívából áll. Az END használata kötelező.** Ezzel a direktívával definiálhatjuk a program belépési pontját vagy kezdőcímét, ha van.

A többi direktíva használata nem kötelező.

Alakja:

END [*kifejezés*]

A kifejezés a **program belépési pontját** határozza meg. A belépési pont az a cím, ahol a program végrehajtása kezdődik. Ha a program egynél több modulból áll, akkor csak egy modulban, a főmodulban lehet belépési pontot definiálni. Ha nem adunk meg belépési pontot, nincs alapfeltételezés.

A makró-assembler az END után álló utasításokat nem veszi figyelembe.

Példa:

```
END kezdet
END
```

5.1.4 A SEGMENT és az ENDS direktíva

A modulokat felépítő szegmensek kijelölésére szolgál.

Alakja:

azonosító **SEGMENT** [*igazítás*][*kombináció*]['*osztálynév*']
azonosító **ENDS**

A SEGMENT direktíva a szegmens elejét, az ENDS a szegmens végét jelzi. A programszegmens utasításokból és/vagy adatokból áll, az utasítások és adatok címe egy adott szegmensregiszterhez viszonyítva értelmezhető.

Az "azonosító" a szegmens neve, amely lehet egyedi, vagy megegyezhet a program más szegmensének nevével. Az egyező nevűekből a makró-assembler egyetlen szegmenst állít elő. Azonos nevű szegmensek esetén mindegyik szegmens-definícióban vagy pontosan azonos attributumokat kell megadni, vagy olyanokat, amelyek nem mondanadk ellent egymásnak.

Az "igazítás", a "kombináció" és az "osztálynév" a program betöltését határozza meg.

A szegmensen belül másik szegmens is elhelyezkedhet, de átlapolásuk nem megengedett.

Az "igazítás" a szegmens kezdőcímének értékét befolyásolja. Igazításként a következő kulcsszavak valamelyikét lehet megadni:

BYTE	a szegmens byte-határon kezdődik
WORD	a szegmens szóhatáron kezdődik (1 szó = 2 byte)
PARA	a szegmens paragrafushatáron kezdődik (1 paragrafus = 16 byte)
PAGE	a szegmens laphatáron kezdődik (1 lap = 1024 byte)

Az igazítás alapértelmezése: PARA

A "kombináció" azt határozza meg, hogy az azonos nevű szegmenseket hogyan állítsa össze a rendszer. A következő esetek lehetségesek:

PUBLIC Az azonos nevű szegmensek egymáshoz fűzésével egyetlen, folytonos szegmenst állít elő. Az új szegmens címei az első ilyen nevű szegmens szegmenscíméhez relatívek. Az összefűzés a szegmensek definiálási sorrendjében történik.

STACK Az azonos nevű szegmensek egymáshoz fűzésével egyetlen, folytonos szegmenst állít elő. A hozzáfűzés a szegmensek definiálási sorrendjében történik. Az új szegmens címei az SS szegmensregiszterhez relatívek. Az SP (Stack Pointer) regiszterbe egy szegmensen belüli cím kerül. Minden programban legalább egy STACK szegmenst definiálni kell. (Kivétel: ha a programból .COM file-t akarunk létrehozni az EXE2BIN DOS paranccsal.)

COMMON Úgy helyezi el az azonos nevű szegmenseket, hogy a kezdőcímük azonos legyen. Az így kialakított terület hossza egyenlő a leghosszabb szegmens hosszával. A szegmensek címei ugyanahhoz a báziscímhez relatívek.

MEMORY Az azonos nevű szegmenseket a memória legnagyobb fizikai szegmensében helyezi el. Ha egynél több MEMORY szegmens van, akkor úgy helyezi el a szegmenseket, mint a COMMON-nál.

AT *cím* A szegmens címei a megadott "cím"-hez relatívek. A "cím" kifejezés is lehet, de nem tartalmazhat olyan szimbólumra való hivatkozást, amelyet később definiálunk a programban. Az AT szegmensek általában nem kódot és adatokat tartalmaznak, hanem címeket, amelyek segítségével memóriabeli kódok és adatok olyanná tehetők, mintha ROM-ban (Read Only Memory) lennének. Az ilyen utasítások és adatok az AT szegmens címkéivel és változóival érhetők el.

Ha nem adunk meg "kombinációt", akkor betöltéskor maga a fizikai szegmens kerül a memóriába.

Az "osztálynév" megadásával a szegmenst az adott nevű osztályba soroljuk. Az egy osztályba tartozó szegmensek folytonos memóriaterületre töltődnek, közöttük nem helyezkedhet el más osztályhoz tartozó szegmens. Az osztálynevet aposztrófok közé kell tenni. Az osztálynév szintaxisa megegyezik az azonosítók szintaxisával.

5.1.5 Az ORG direktíva

Alakja:

ORG *kifejezés*

Az ORG direktíva hatására a "kifejezés" értéke az utasításszámlálóba kerül. Ez lesz a következő utasítás vagy adat címe az aktuális szegmensen belül. A kifejezés értékének egy konkrét számnak kell lennie, vagyis a kifejezésben szereplő minden szimbólumnak ismertnek kell lennie a fordítás első menetében. A kifejezésben a számlálóra hivatkozó szimbólum ($) is előfordulhat.

(Megjegyzés: ha programunkból .COM file-t akarunk létrehozni, a szegmensben meg kell adni az ORG 100H direktívát.)

5.1.6 Az EVEN direktíva

Alakja:

EVEN

Az EVEN direktíva az azt követő adatot vagy utasítást szóhatárra igazítja. Ha a számláló aktuális értéke páratlan, akkor a direktíva hatására egy NOP (üres) utasítást (1 byte-ot) generál a fordító, így a számláló értéke 1-gyel nő és páros lesz. Ha a számláló értéke páros, akkor az EVEN direktíva hatástalan. Byte-határra igazított szegmensekben nem lehet az EVEN direktívát használni.

5.1.7 A PROC és az ENDP direktíva

Ezt a két direktívát eljárások deklarálására használjuk.

Alakja:

név **PROC** [*típus*]
utasítások
név **ENDP**

A PROC direktíva az eljárás kezdetét, az ENDP direktíva a végét jelzi. Az eljárás nevének egyértelműen kell azonosítania az eljárást. A név lehet JMP, CALL és LOOP utasítás operandusa is. Az eljárás típusát a NEAR vagy a FAR kulcsszóval határozhatjuk meg. Ha a típust nem adjuk meg, akkor NEAR az alapértelmezés.

A PROC és az ENDP direktívák között tetszőleges számú utasítás helyezkedhet el. Minden eljárásnak legalább egy RET utasítást kell tartalmaznia, amelynek hatására visszakerül a vezérlés a hívás helyére.

Az eljárások nem paraméterezhetők.

5.1.8 A GROUP direktíva

A GROUP direktívát szegmenscsoportok kialakítására használjuk.

Alakja:

azonosító **GROUP** *szegmensnév*,,,

A GROUP direktívával egy vagy több szegmensből álló csoportot definiálhatunk. Az "azonosító" a szegmenscsoport nevét jelenti. A szegmenscsoport címkéinek és változóinak címe a csoport elejéhez relatív, nem annak a szegmensnek a kezdetéhez, amelyben definiálták. A szegmenscsoport nevének egyedinek kell lennie. A "szegmensnév" olyan azonosító, amelyet SEGMENT direktívában vagy SEG kifejezésben definiáltunk.

Az egy szegmenscsoportba tartozó különféle szegmensek az elsőként megadott szegmens szegmenscíméhez relatívek, így a csoport objektumainak címzése egyszerűbb.

A GROUP direktíva nem határozza meg a csoport szegmenseinek betöltési sorrendjét. A betöltési sorrend vagy a szegmensek osztályától függ, vagy attól, hogy a tárgymodulok milyen sorrendben kerülnek a futtatható programot előállító szerkesztőhöz.

A csoport szegmenseinek nem kötelező folytonosan elhelyezkedniük, vagyis a szegmensek közé olyan szegmens is betölthető, amely nem tartozik a csoporthoz. Figyelembe kell azonban venni, hogy a csoport első szegmensének első byte-ja és a csoport utolsó szegmensének utolsó byte-ja közötti távolság nem haladhatja meg a 65535 byte-ot. Ha a csoport szegmensei folyamatosan helyezkednek el, akkor a csoport mérete elérheti a 64 Kbyte-ot.

A csoportneveket ASSUME direktívában (ld. 1.9) és a címkiszámítás viszonyítási alapjának megváltoztatására lehet használni.

A forrás-file-ban egy adott csoportnevet csak egyetlen GROUP direktívában lehet használni. **Ha a csoport több szegmensből áll, akkor a csoporthoz tartozó összes szegmens nevét ugyanabban a GROUP direktívában kell megadni.**

5.1.9 Az ASSUME direktíva

Az ASSUME direktívával adott szegmensekhez adott szegmens-regisztereket rendelünk.

Alakja:

ASSUME *szegmens-regiszter:szegmensnév*,,,
ASSUME NOTHING

Az ASSUME direktíva a "szegmensregiszter"-hez hozzárendeli a "szegmensnév"-vel megadott szegmens szegmenscímét. Ezután a fordító minden címképzési folyamatban az objektumok címét a címképzési módhoz tartozó szegmensregiszterhez rendelt szegmens szegmenscíme alapján számítja ki.

Szegmensregiszterként a CS, DS, ES vagy SS regiszter adható meg.

Az ASSUME direktívával biztosítjuk, hogy a szegmens objektumai elérhetők az adott szegmensregiszterrel. Ennek megvalósulásához azonban előzőleg a programozónak gondoskodnia kell a szegmensregiszter értékének beállításáról is!

A szegmensnév a következők valamelyike lehet:

- SEGMENT direktívával már előzőleg definiált szegmens azonosítója
- GROUP direktívával már előzőleg definiált csoport azonosítója
- a NOTHING kulcsszó.

A direktíva ASSUME NOTHING alakjának hatására az összes előző szegmenskijelölés megszűnik.

A címkiszámítás viszonyítási alapját megváltoztató műveletet (:) felhasználhatjuk az ASSUME direktívával kijelölt szegmens megváltoztatására.

5.1.10 A megjegyzések és a COMMENT direktíva

A megjegyzések tetszőleges karakterekből állhatnak. Bárhol elhelyezhetők a programban. A megjegyzéseket a fordító nem veszi figyelembe. Az egy sornál nem hosszabb megjegyzések alakja:

;szöveg

Az egy sornál hosszabb megjegyzések írására a COMMENT direktívát használjuk.

Alakja:

COMMENT *határoló szöveg határoló*

A COMMENT direktíva hatására a fordító megjegyzésnek tekinti a határoló jelek közötti szöveget. A határoló jel a COMMENT kulcsszó után következő első olyan karakter, amely nem szóköz. A megjegyzés a határoló jel legközelebbi előfordulásáig tart. *A második határoló jel sorában a határoló jel utáni karaktereket sem veszi figyelembe a fordító.*

5.1.11 Példák a makró-assembly programok felépítését meghatározó direktívák használatára

A mintaprogram két modulból áll, az első az 'indító', a második a 'kiíró' modul. Ezeknek természetesen két forrásfile-ban kell lenniük (INDITO.ASM és KIIRO.ASM). Mindegyik modulban több szegmens van, az első modul a főprogram, tehát ebben adjuk meg a program belépési (indítási) pontját (ez a 'start' címke). Indulás után a program beállítja az adatszegmensregisztert, majd a második modulban levő kiírás címkére adja a vezérlést, ami DOS hívás segítségével kiírja a 'szöveg'-et, majd visszaadja a vezérlést az első modul 'vége' címkéjére, amely visszatér a DOS-hoz.

Figyeljük meg, hogy mindkét modulban az elérni kívánt szegmensek deklarációja PUBLIC, a másik modul által használt címkék szintén PUBLIC-ként vannak kijelölve, a másik modulbol hívott címke pedig külső (EXTRN) megjelölést kap. Mindket modul az END direktívával zárul, de csak a főprogramnak szánt 'indító' modulban adjuk meg az indítócímet.

Az INDITO.ASM file listája:

```
        NAME    indito
        PUBLIC  start, vege
        EXTRN   kiiras:near
stack   SEGMENT WORD PUBLIC 'STACK'
        dw      64 DUP(?)
stack   ENDS
data    SEGMENT WORD PUBLIC 'DATA'
data    ENDS
code    SEGMENT BYTE PUBLIC 'CODE'
        ASSUME  cs:code,ds:data
start:  mov     ax, seg DATA
        mov     ds, ax
        jmp     kiiras
vege:   mov     ax, 4C00H
        int     21H
code    ENDS
        END     start
```

A KIIRO.ASM file listája:

```
        NAME    kiiro
        PUBLIC  kiiras
        EXTRN   vege:near
data    SEGMENT WORD PUBLIC 'DATA'
szoveg  db      'Mintaprogram'
        db      13,0AH,'$'
data    ENDS
code    SEGMENT BYTE PUBLIC 'CODE'
        ASSUME  cs:code,ds:data
```

```
kiiras: mov     dx, offset szoveg
        mov     ah, 9
        int     21H
        jmp     vege
code    ENDS
        END
```

A két modulból a következő módon kell futtatható programot készíteni:

1. Fordítsuk le az első modult:
   C > MASM INDITO,,;
2. Fordítsuk le a második modult:
   C > MASM KIIRO,,;
3. Szerkesszük össze a két modult futtatható programmá:
   C > LINK INDITO + KIIRO;

Az 1. lépés hatására egy
INDITO.OBJ és egy
INDITO.LST

a 2. lépés hatására egy
KIIRO.OBJ és egy
KIIRO.LST
a 3. lépés után pedig egy
INDITO.EXE

file keletkezik a C meghajtón. Az LST file-ok a modulok listait, az OBJ-ok a legfordított programot, az EXE file pedig az összeszerkesztett kész programot tartalmazzák.

Ezután írjuk be az

C > INDITO < Enter >

parancsot és a képernyőn megjelenik a

```
Mintaprogram
```

szöveg, amit programunk írt ki.

5.2 Deklarációk

Ebben a fejezetben azt írjuk le, hogyan lehet adatokat előállítani a program számára, hogyan lehet az utasításokra és adatokra vonatkozó címkéket, változókat és egyéb szimbólumokat deklarálni, valamint hogyan definiálhatók a több mezőből álló struktúrák és rekordok.

5.2.1 Adatdeklarációk

Az adatdeklarációk lehetővé teszik a programozó számára, hogy programjához adatokat definiáljon. A direktívák byte-okra, szavakra vagy egyéb memóriaegységekre képezik le a

számokat, karaktersorozatokat és kifejezéseket. Az adatok bekerülnek a program tárgykódú változatába, a program object file-jába.

A következő adatdeklaráló direktívák adhatók meg:

DB	byte
DW	szó (= 2 byte)
DD	dupla szó
DQ	négy szó
DT	10 byte-os összefüggő terület

A DB direktíva (Define Byte)

Alakja:

[*azonosító*] **DB** *kezdőérték*[„„]

A DB direktíva konstansok, vagy (kezdőértékkel ellátott) változók deklarálására alkalmas. A konstans vagy a változó kezdőértéke egész szám, karaktersorozat, DUP művelet (ld. ebben a fejezetben később), konstans kifejezés vagy kérdőjel lehet. A kérdőjel azt jelenti, hogy a kezdőérték definiálatlan. Két vagy több kezdőérték megadása esetén az értékeket vesszővel kell elválasztani egymástól. A DB direktíva a karaktersorozat kivételével minden kezdőérték számára 1 byte-ot foglal le. A karaktersorozat kezdőértékhez a direktíva annyi byte-ot rendel, amennyi a sorozat karaktereinek száma.

Az "azonosító" megadása nem kötelező. Ha megadjuk, akkor a DB direktíva egy olyan BYTE típusú változót állít elő, amelynek relatív címe a számláló aktuális értékével egyenlő. (Byte típusú változó: a változóra való hivatkozáskor 1 byte-nyi területre hivatkozunk.)

A karaktersorozat-konstans legfeljebb annyi karakterből állhat, amennyi az adott DB direktíva sorában elfér. A karakterek a felírási sorrendben tárolódnak, vagyis az első karakter címe a legkisebb és az utolsóé a legnagyobb.

Egyetlen DB direktívával különböző típusú kezdőértékek megadhatók, de ekkor a kezdőértékek között nem szerepelhet DUP művelet.

Példák:

```
egész              DB 3
karaktersorozat    DB 'MUNKAHELYE:'
vektor             DB 5 DUP(0)
konstans-kifejezés DB 10*10
definiálatlan      DB ?
vegyes             DB 6, 4*2, 'SZOVEG', ?
```

A DW direktíva (Define Word)

Alakja:

[*azonosító*] **DW** *kezdőérték*[„„]

A DW direktíva konstansok vagy (kezdőértékkel ellátott) változók deklarálására alkalmas. A konstans vagy a változó kezdőértéke egész szám, karaktersorozat, DUP

művelet, konstans kifejezés, címkifejezés vagy kérdőjel lehet. A kérdőjel definiálatlan kezdőértéket jelent.

Ha az opcionális "azonosító"-t megadjuk, akkor a DW direktíva egy olyan WORD típusú változót állít elő, amelynek relatív címe a számláló aktuális értékével egyenlő. (WORD típusú változó: a változóra való hivatkozáskor 1 szónyi területre hivatkozunk.)

A DW direktíva minden megadott kezdőérték számára egy-egy szót (1 szó = 2 byte) foglal le. A karaktersorozat-konstansok legfeljebb 2 karakter hosszúságúak lehetnek. Az utolsó vagy egyetlen karakter az alacsonyabb helyiértékű byte-on, az első karakter a magasabb helyiértékű byte-on tárolódik. Ha a string egyetlen karakterből áll, akkor a magasabb helyiértékű byte-ra nulla kerül.

Egyetlen DW direktívával különböző típusú kezdőértékek megadhatók, de ekkor a kezdőértékek között nem szerepelhet DUP művelet.

Példák:

```
egész               DW 318
karaktersorozat     DW 'ab'
vektor              DW 5 DUP(0)
konstans-kifejezés  DW 10*7
címkifejezés        DW egész
definiálatlan       DW ?
vegyes              DW 512, 2*3, 'cd', ?
```

A DD direktíva (Define Doubleword)

Alakja:

[*azonosító*] **DD** *kezdőérték*[,,,]

A DD direktíva minden megadott "kezdőérték" számára egy-egy dupla szavas (4 byte-os) tárterületet foglal le, és beállítja az adott kezdőértékre. A "kezdőérték" egész szám, valós szám, karaktersorozat-konstans, kódolt valós szám, DUP művelet, konstans kifejezés, címkifejezés vagy kérdőjel lehet. A kérdőjel definiálatlan kezdőértéket jelent.

Ha az opcionális "azonosító"-t megadjuk, akkor a DD direktíva egy olyan DWORD típusú változót állít elő, amelynek relatív címe a számláló aktuális értékével egyenlő. (DWORD típusú változó: a változóra való hivatkozáskor egy dupla szónyi tárterületre hivatkozunk.)

A karaktersorozat-konstansok legfeljebb 2 karakter hosszúságúak lehetnek. Ha egy karakterünk van, az a legalacsonyabb helyiértékű byte-ba kerül, a többi byte 0-val töltődik fel, ha kettő, akkor abból a másodiknak leírt kerül a legalacsonyabb helyiértékű byte-ra, az elsőnek leírt az ez utáni byte-ra, a többi byte 0 lesz.

Egyetlen DD direktívával különböző típusú kezdőértékek megadhatók, de ekkor a kezdőértékek között nem szerepelhet DUP művelet.

Példák:

```
egész               DD 318
```

```
valós              DD 31.8
karaktersorozat    DD 'ab'
kódolt-valós       DD 3F800000R
vektor             DD 5 DUP(0)
konstans-kifejezés DD 10*7
címkifejezés       DD egész
definiálatlan      DD ?
vegyes             DD 622, 42.3, 'cd', kódolt-valós, ?
```

A DQ direktíva (Define Quadword)

Alakja:

[*azonosító*] **DQ** *kezdőérték*[,,,]

A DQ direktíva konstansok vagy (kezdőértékkel ellátott) változók deklarálására alkalmas. A konstans vagy a változó kezdőértéke egész szám, valós szám, karaktersorozat, kódolt valós szám (ld. II. fejezet), DUP művelet, konstans kifejezés vagy kérdőjel lehet. A kérdőjel definiálatlan kezdőértéket jelent.

Ha az opcionális azonosítót megadjuk, akkor a DQ direktíva egy olyan QWORD típusú változót állít elő, amelynek relatív címe a számláló aktuális értékével egyenlő. (QWORD típusú változó: a változóra való hivatkozáskor egy 4 szónyi tárterületre hivatkozunk.)

A DQ direktíva minden megadott kezdőérték számára 4 szót foglal le.

A karaktersorozat-konstansok legfeljebb 2 karakter hosszúságúak lehetnek. Ha egy karakterünk van, az a legalacsonyabb helyiértékű byte-ra kerül, a többi byte 0-val töltődik fel, ha kettő, akkor abból a másodiknak leírt kerül a legalacsonzabb helyiértékű byte-ra, az elsőnek leírt az ez előtti byte-ra, a többi byte 0 lesz.

Egyetlen DQ direktívával különböző típusú kezdőértékek megadhatók, de ekkor a kezdőértékek között nem szerepelhet DUP művelet.

Példák:

```
egész              DQ 318
valós              DQ 31.8
karaktersorozat    DQ 'ab'
kódolt-valós       DQ 3F80000000000000R
vektor             DQ 5 DUP(0)
konstans-kifejezés DQ 10*7
definiálatlan      DQ ?
vegyes             DQ 512, 44.6, 'cd', 2*3, ?
```

A DT direktíva (Define Tenbytes)

Alakja:

[*azonosító*] **DT** *kezdőérték*[,,,]

A DT direktíva konstansok vagy (kezdőértékkel ellátott) változók deklarálására alkalmas. A konstans vagy a változó kezdőértéke egész kifejezés, tömörített decimális szám, karaktersorozat-konstans, kódolt valós szám, DUP művelet vagy kérdőjel lehet. A kérdőjel definiálatlan kezdőértéket jelent.

Ha az opcionális azonosítót megadjuk, akkor a DT direktíva egy olyan TBYTE típusú változót állít elő, amelynek relatív címe a számláló aktuális értékével egyenlő. (TBYTE típusú változó: a változóra való hivatkozáskor egy 10 byte-nyi tárterületre hivatkozunk.)

A DT direktíva minden megadott kezdőérték számára egy 10 byte-os területet foglal le.

A karaktersorozat-konstansok legfeljebb 2 karakter hosszúságúak lehetnek. Ha egy karakterünk van, az a legalacsonyabb helyiértékű byte-ra kerül, a többi byte 0-val töltődik fel, ha kettő, akkor abból a másodiknak leírt kerül a legalacsonyabb helyiértékű byte-ra, az elsőnek leírt az ez utáni byte-ra, a többi byte 0 lesz.

A DT direktíva alapfeltételezése, hogy a decimális konstansok tömörített decimálisak, nem egész számok. Ezért a decimális egészek után "D"-t kell tenni.

Egyetlen DT direktívával különböző típusú kezdőértékek megadhatók, de ekkor a kezdőértékek között nem szerepelhet DUP művelet.

Példák:

```
egész                DT 318D
tömörített-decimális DT 318
karaktersorozat      DT 'ab'
kódolt-valós         DT 3F80000000000000000R
vektor               DT 5 DUP(0)
definiálatlan        DT ?
vegyes               DT 193D, 'cd', ?
```

A DUP művelet

Felírási módja:

szám **DUP**(*kezdőérték*,,,)

A DUP műveleti jelet olyankor használjuk, amikor egy vagy több kezdőértéket ismételni akarunk. A "szám" határozza meg, hogy hányszor ismétlődjön a kezdőérték. A kezdőérték lehet kérdőjel karakter (határozatlan kezdőértéket jelent), vagy egy olyan kifejezés, amelynek értéke egy egész szám, karakterkonstans vagy egy másik DUP művelet. *A DUP műveletek 17 szint mélységig ágyazhatók egymásba.* Ha egynél több kezdőértéket adunk meg, akkor a kezdőértékeket vesszővel kell elválasztani egymástól. Ilyenkor a kezdőértékek sorozatonként ismétlődnek.

Példák:

```
DB 10 DUP(10 DUP(1))
```

Ez a direktíva 100 byte-ot foglal le és mindegyik byte az 1 értéket tartalmazza.

```
DW 4 DUP(3 DUP(2 DUP(9,8,7)))
```

Ez a direktíva 24 szót foglal le és a szavak rendre a 9,8,7,9,8,7,... értéket tartalmazzák.

DB 20 DUP(?)

A direktíva hatására a fordító 20 byte-ot foglal le és a byte-ok tartalma definiálatlan.

DB 3 DUP(2 DUP(4,6))

Ez a direktíva 6 byte-ot foglal le, amelyek értéke a legkisebb című byte-tól a legnagyobb című byte felé haladva: 4,6,4,6,4,6

5.2.2 Címkedeklarációk

A címkedeklarációval olyan azonosítót definiálhatunk, amellyel egy adott utasítás címére hivatkozhatunk. A címkéket a program vezérlésátadó utasításaiban használhatjuk (JMP, CALL, LOOP utasítás).

A címkék NEAR vagy FAR típusúak lehetnek. A NEAR típusú címkére csak abban a szegmensben hivatkozhatunk, amelyikben a címkét definiáltuk. A FAR típusú címkére a program bármely szegmensében hivatkozhatunk.

NEAR típusú címkéket kétféleképpen deklarálhatunk:

azonosító: [*utasítás*]

vagy

azonosító **PROC NEAR**

megadásával.

FAR típusú címkét a következőképpen definiálhatunk:

azonosító **PROC FAR**

Egy adott NEAR típusú címkét egy szegmensen belül csak egyszer, egy FAR típusú címkét pedig egy programon belül csak egyszer definiálhatunk.

A címke a deklarációját követő első utasítás címét jelenti, értéke az utasításszámláló aktuális értékével lesz egyenlő.

A címke a deklarációját tartalmazó szegmenshez kijelölt szegmensregiszterhez (ld. 5. fejezet 1.9 pont ASSUME direktíva) relatív.

Címkét a LABEL és a THIS direktívával is deklarálhatunk, ezek ismertetése e fejezetben található.

5.2.3 Szimbólumdeklarációk

A szimbólumdeklaráló direktívák lehetővé teszik a programozó számára, hogy szimbólumokat definiáljon és használjon. A szimbólum egy olyan azonosító, amelynek értéke egy szám, egy szöveg, egy utasítás vagy egy cím lehet. A szimbólumok használata olvashatóbbá teszi a programot. A szimbólumra bármely olyan helyen hivatkozhatunk, ahol a szimbólumhoz rendelt érték használata megengedett.

A következő szimbólumdeklaráló direktívák adhatók meg:

= szimbólum hozzárendelése egy 16-bites értékhez

EQU	szimbólum hozzárendelése egy 16-bites értékhez, egy karaktersorozat-konstanshoz vagy karakterek egyéb kombinációjához, szimbólumok átnevezése
LABEL	adott típusú változó vagy címke definiálása

Az = direktíva

Alakja:

azonosító = kifejezés

Az = direktíva hozzárendeli az "azonosító"-val jelölt szimbólumhoz a "kifejezés" numerikus értékét. Az azonosítóval 16-bites értékre hivatkozhatunk. A direktíva hatására nem történik helyfoglalás. Az azonosítóra való hivatkozáskor a fordító mindig a kifejezés értékét helyettesíti be a szimbólum helyére.

A "kifejezés" egész szám, 1 vagy 2 karakterből álló karaktersorozat-konstans, konstans-kifejezés vagy címkifejezés lehet. A kifejezés értéke nem lehet nagyobb, mint 65535. Az "azonosító" vagy olyan szimbólum, amelyet még nem definiáltunk, vagy olyan, amelyet korábban egy = direktívával deklaráltunk, az = direktívával deklarált szimbólumok ugyanis **újradefiniálhatók**.

Példák:

egész	= 8364
karaktersorozat	= 'ab'
konstans-kifejezés	= 5*10
címkifejezés	= egész

Az EQU direktíva

Alakja:

azonosító **EQU** *kifejezés*

Az EQU direktíva hozzárendeli az "azonosító"-val jelölt szimbólumhoz a "kifejezést". A szimbólummal egy 16-bites értékre, egy karaktersorozatra vagy karakterek egyéb kombinációjára hivatkozhatunk. Az EQU direktívát szimbólumok átnevezésére is lehet használni.

A fordító az "azonosító" minden további előfordulását egy szöveggel vagy a "kifejezés" értékével helyettesíti, attól függően, hogy milyen típusú az EQU direktívában megadott kifejezés.

Az azonosítónak olyan szimbólumnak kell lennie, amelyet előzőleg még nem definiáltunk. A "kifejezés" egész szám, karaktersorozat-konstans, valós szám, kódolt valós szám, utasítás mnemonikus kódja, konstans kifejezés vagy címkifejezés lehet. Az olyan kifejezéssel, amelynek értéke egy [0,65535] intervallumba eső egész szám, egy 16-bites értéket reprezentáló szimbólumot deklatálhatunk. Ekkor az "azonosító" minden előfordulását a megadott értékkel helyettesíti a fordító. Bármely más kifejezés megadása esetén az "azonosító"-val egy szöveg típusú szimbólumot deklarálunk, így az "azonosító" helyébe az adott szöveg kerül.

Az EQU direktívával deklarált szimbólumokat nem lehet újradefiniálni.

Példák:

```
egész                   EQU 8364
karaktersorozat-konstans EQU 'üzenet'
valós                   EQU 2.5
mnemonik                EQU ADD
konstans-kifejezés      EQU 5*10
címkifejezé             EQU egész
```

A LABEL direktíva

Alakja:

azonosító **LABEL** *típus*

A LABEL direktíva lehetővé teszi, hogy a memória valamely területére különböző típusú azonosítókkal hivatkozhassunk, pl. egy duplaszó (DWORD típus) első byte-jára (BYTE típusként) és így tovább. A LABEL direktíva hatására nem történik helyfoglalás. A fordító hozzárendeli az azonosítóhoz a számláló aktuális értékét és a direktívában megadott típust. *Az azonosítónak olyan szimbólumnak kell lennie, amelyet előzőleg még nem definiáltunk.*

Típusként a következő kulcsszavak valamelyike adható meg:

változók típusa
- BYTE
- WORD
- DWORD
- QWORD
- TBYTE

címkék típusa
- NEAR
- FAR

A FAR típusú utasításcímkéket kötelező LABEL vagy PROC direktívával deklarálni, mert a fordító alapfeltételezése a címkék típusára vonatkozóan NEAR.

Példa:

```
a LABEL  BYTE
b DD ?
```

A programban a "b" változóval egy duplaszóra hivatkozhatunk, az "a" változóval pedig ennek a duplaszónak az első byte-jára.

5.2.4 Struktúrák deklarálása

A struktúrák olyan adatterületek, amelyek több, különböző típusú adatból állnak. Deklarálásuk két lépésből áll: először a STRUC, ENDS direktívapár segítségével a

struktúra szerkezetét deklaráljuk, azután a struktúra elemeinek kezdőértékét. Az első lépést **struktúra-típusdeklarációnak**, a másodikat **struktúra-deklarációnak** nevezzük.

Struktúra-típusdeklaráció (a STRUC és az ENDS direktíva)

A struktúra típus deklarálása:

struktúranév **STRUC**
Mező-definíciók
struktúranév **ENDS**

A STRUC direktíva a struktúra típusdeklarációjának a kezdetét, az ENDS direktíva pedig a végét jelenti. A "struktúranév" a struktúra-típus neve, amely egyértelműen azonosítja a struktúra-típust. A mező-definíciók a struktúra-típus mezőinek számát, a mezők típusát és alapfeltételezés szerinti értékét határozzák meg. Tetszőleges számú mező-definíció megadható. Egy mező-definíció a következők valamelyike lehet:

[*mezőnév*] **DB** *alapfeltételezés*,,,
[*mezőnév*] **DW** *alapfeltételezés*,,,
[*mezőnév*] **DD** *alapfeltételezés*,,,
[*mezőnév*] **DQ** *alapfeltételezés*,,,
[*mezőnév*] **DT** *alapfeltételezés*,,,

Az opcionális "mezőnév" a mező neve, amelynek egyértelműnek kell lennie. A mezőnév az adott mezőnek a struktúra kezdetétől való távolságát reprezentálja. A DB, DW, DD, DQ és DT direktíva a mező méretét határozza meg.

Egy, a programban már definiált struktúra-típus felhasználható struktúra-változó deklarálásához. Ha a struktúra-változó deklarálásakor valamely mezőhöz nem adunk meg kezdőértéket, akkor a mező értéke a struktúra-típus deklarációban megadott alapfeltételezéssel lesz egyenlő.

Az "alapfeltételezés" egy szám, karakter- vagy karaktersorozat-konstans vagy szimbólum lehet és DUP műveletet is tartalmazhhat. Ha az alapfeltételezést karaktersorozat-konstanssal adjuk meg, akkor a mező annyi byte hosszúságú lesz, ahány karakterből áll a string. A struktúra-típus deklarációja kizárólag mező-definíciókat és megjegyzéseket tartalmazhat, vagyis a struktúrák nem ágyazhatók egymásba.

Példa:

```
s       STRUC
mező1   DB 'string'
mező2   DB 5
mező3   DW 3 DUP(?)
s       ENDS
```

Struktúra-deklaráció

A struktúra-deklarációk segítségével egy előzőleg már definiált struktúra-típus mezőinek kezdőértéket adhatunk. A struktúra-deklarációk alakja:

[*azonosító*] *struktúranév* < [*kezdőérték*],,, >

Az *"azonosító"* egy struktúraváltozó neve. A struktúraváltozó egy vagy több mezőből állhat. A különböző mezők különböző méretűek lehetnek. Az azonosítót nem kötelező megadni. Ha elhagyjuk, akkor a fordító területet foglal le a struktúra számára, de nem állít elő olyan azonosítót, amelynek segítségével hozzá lehetne férni a struktúrához.

A *"struktúranév"* egy korábban STRUC direktívával definiált struktúra típus azonosítója.

A *"kezdőértékek"* a struktúra elemeinek kezdőértékét határozzák meg. Minden mezőhöz legfeljebb egy kezdőérték adható meg. A kezdőérték egész szám, karaktersorozat-konstans vagy olyan kifejezés lehet, amely értékének típusa megegyezik a megfelelő mező típusával. A csúcsos zárójelpár (< >) használata akkor is kötelező, ha egyetlen kezdőértéket sem adunk meg. Ha egynél több kezdőérték szerepel a deklarációban, akkor a kezdőértékeket vesszővel kell elválasztani egymástól. A DUP művelet használata esetén csak a kerek zárójelek közötti értékeket kell csúcsos zárójelek közé tenni.

A kezdőértékek megadása nem kötelező. Ha valamely struktúramező számára nem deklarálunk kezdőértéket, akkor a fordító automatikusan a struktúra típus definiálásakor (STRUC direktíva) megadott alapfeltételezés szerinti értéket veszi figyelembe. Ha nincs alapfeltételezés szerinti érték sem, akkor a mező értéke definiálatlan.

A struktúra-deklarációban nem lehet kezdőértéket adni az olyan mezőknek, amelyekhez a STRUC direktívában több alapfeltételezés szerinti értéket rendeltünk.

Példák:

```
str1 s < >
```

A "str1" struktúraváltozót állítja elő, amelynek típusát az "s" struktúra típussal deklaráltuk. A struktúramezők kezdőértékei megegyeznek a struktúra típus deklarációjában megadott alapfeltételezés szerinti értékekkel, ha ezek léteznek.

```
str2  s  <,2,>
```

A "str2" struktúraváltozót állítja elő, amely "s" típusú. Az első és a harmadik mező kezdőértéke megegyezik a struktúra típus deklarációjában megadott alapfeltételezés szerinti értékkel. A második mező kezdőértéke 2-re állítódik be.

```
str3  s  5 DUP(<,2,>)
```

A "str3" struktúraváltozót állítja elő, amely 5 darab "s" típusú struktúrából áll. Minden ilyen struktúra első és harmadik mezőjének kezdőértéke megegyezik a struktúra típus deklarálásakor megadott alapfeltételezés szerinti értékkel, és minden ilyen struktúra második mezőjének kezdőértéke 2-re állítódik be.

Példa a struktúra típusdeklaráció és a struktúra-deklaráció együttes használatára:

```
struk  STRUC
a      DB  ?
b      DB  'darab'
c      DB  'ára (\)'
d      DW  ?
struk  ENDS
struktúra struk <23,,,2300>
```

A STRUC, ENDS direktívapárral és a köztük lévő DB, DW direktívákkal a "struk" struktúra szerkezetét és a struktúra-elemek alapfeltételezés szerinti kezdőértékét deklaráltuk. A struktúra-típus deklaráció után a "struktúra" struktúra-változó deklarációja következik, amely "struk" típusú struktúra. Az első struktúra-elem kezdőértékét 23-ra állítottuk be, a második és harmadik mező kezdőértéke megegyezik a struktúra típusdeklarációjában megadott megfelelő kezdőértékkel, a negyedik mező kezdőértéke 2300.

5.2.5 Rekordok deklarálása

A rekord olyan egy byte vagy egy szó nagyságú terület, amely egy vagy több bitmezőből áll. Deklarálásuk két lépésből áll: először a RECORD direktívával a rekord szerkezetét deklaráljuk, azután a rekord bitmezőinek kezdőértékét. Az első lépést rekord-típus deklarációnak, a másodikat rekord-deklarációnak nevezzük.

Rekord-típus deklaráció (a RECORD direktíva)

A rekord típus definiálása:

rekordnév **RECORD** *mezőnév*:*szélesség*[= *kifejezés*][,,,]

A RECORD direktívával egy 8- vagy 16-bites rekordot definiálhatunk, amely egy vagy több mezőből áll.

A "*rekordnév*" annak a rekordtípusnak az azonosítója, amelyet rekord előállításakor használunk. A "*mezőnév*" egy rekordbeli mező azonosítója, amelynek bitekben mért hosszát a "*szélesség*" határozza meg. A "*kifejezés*" a mező alapfeltételezés szerinti értéke. **Ha több mezőt adunk meg, ügyeljünk arra, hogy a mezőszélességek összege ne haladja meg a 16-ot!**

A szélességnek olyan konstansnak kell lennie, amelyre: 1< = szélesség < = 16. Ha a deklarált mezők szélességeinek összege meghaladja a 8 bitet, akkor a rekord 2 byte hosszúságú, egyébként 1 byte hosszúságú lesz.

Ha a mező legalább 7-bites, akkor alapfeltételezés szerinti értékként egy ASCII karakter is megadható.

A kifejezésben csak már definiált adatokra lehet hivatkozni.

A bitmezők elhelyezése a rekordban a felsorolás sorrendjében balról jobbra történik. Ha a deklarált mezők nem töltik ki teljesen a 8 vagy a 16 bitet, akkor az egész rekord jobbra tolódik úgy, hogy az utolsó mező utolsó bitje lesz a rekord legalacsonyabb helyiértékű bitje. Ilyenkor a rekord legmagasabb helyiértékű bitjei egyik mezőhöz sem tartoznak.

Példa:

```
bitmezők RECORD x:2, y:4 = 1, z:7 = 'A'
```

Rekord-deklarációk

A rekord-deklarációk alakja:

[*azonosító*] *rekordnév* < [*kezdőérték*],,, >

A rekordváltozó egy 8- vagy 16-bites érték, amely egy vagy több bitmezőből áll. Az "*azonosító*" a rekordváltozó neve, amelyet nem kötelező megadni. Ha elhagyjuk, akkor a fordító területet allokál a rekord számára, de nem állít elő olyan azonosítót, amellyel a rekordot el lehetne érni.

A "*rekordnév*" egy korábban RECORD direktívával megadott rekord típus azonosítója.

A "*kezdőértékek*" a rekord mezőinek kezdőértékét határozzák meg. Minden mezőhöz legfeljebb egy kezdőérték adható meg. A kezdőérték egész szám, karaktersorozat-konstans vagy olyan kifejezés lehet, amelynek értéke ábrázolható az adott hosszúságú mezőben. A csúcsos zárójelek használata akkor is kötelező, ha egyetlen kezdőértéket sem adunk meg. Ha egynél több kezdőérték szerepel a deklarációban, akkor a kezdőértékeket vesszővel kell elválasztani egymástól.

A DUP művelet használata esetén csak a kerek zárójelek közötti értékeket kell csúcsos zárójelek közé tenni.

A kezdőértékek megadása nem kötelező. Ha valamely rekordmező számára nem deklarálunk kezdőértéket, akkor a fordító automatikusan a rekord típus definiálásakor (RECORD direktíva) megadott alapfeltételezés szerinti értéket veszi figyelembe. Ha nincs alapfeltételezés szerinti érték sem, akkor a mező értéke definiálatlan.

Példák:

r1 bitmezők < >

Az "r1" rekordváltozót állítja elő, amelynek típusát a "bitmezők" rekord-típussal deklaráltuk. A rekordmezők kezdőértékei megegyeznek a rekord típus deklarációjában megadott alapfeltételezés szerinti értékekkel, ha ezek léteznek.

r2 bitmezők <,2, >

Az "r2" rekordváltozót állítja elő, amely "bitmező" típusú. Az első és a harmadik mező kezdőértéke megegyezik a rekord-típus deklarációjaban megadott alapfeltételezés szerinti értékkel. A második mező kezdőértéke 2-re állítódik be.

r3 bitmezők 5 DUP(<,2,'B' >)

Az "r3" rekordváltozót állítja elő, amely 5 darab "bitmezők" típusú rekordot tartalmaz. Minden rekord első mezőjének kezdőértéke megegyezik a rekord-típus deklarációjában megadott alapfeltételezés szerinti értékkel. A rekordok második mezőjének kezdőértéke 2-re, a harmadiké 'B'-re állítódik be.

Példa a rekord-típus deklaráció és a rekord-deklaráció együttes használtára:

```
rek RECORD bm1:5 = 8, bm2:4, bm3:7 = 'B'
rekord rek <3,5, >
```

A RECORD direktívával a "rek" rekord szerkezetét és a rekord bitmezőinek alapfeltételezés szerinti kezdőértékét deklaráltuk. A "bm1" bitmező 5-bites, a "bm2" 4-bites, "bm3" 7-bites. "bm1" alapfeltételezés szerinti kezdőértéke 8, "bm2"-nek nincs

alapfeltételezés szerinti kezdőértéke, a "bm3" alapfeltételezés szerinti kezdőértéke a 'B' karakter.

A rekord típusdeklaráció után a "rekord" rekordváltozó deklarációja következik, amellyel az első bitmező értékét 3-ra, a második bitmező értékét 5-re állítjuk be. A harmadik bitmező értéke továbbra is a 'B' karakter marad.

5.2.6 Előre-hivatkozások

A macro-assembler megengedi, hogy cimkére, változóra és egyéb szimbólumra a deklaráló utasítást megelőzve (előre) hivatkozzunk.

Például:

```
JMP cimke
ADD ...
.
.
.
cimke:
```

Az előre-hivatkozások helytelen használata fordítási hibákhoz vezethet. Ha a fordító definiálatlan azonosítót talál az első fordítási menetben, akkor előre hivatkozást feltételez. Ha csak az azonosítót adjuk meg, akkor a fordító az alapfeltételezés szerinti típus és szegmensregiszter alapján generálja az utasítás kódját és adatait. Az alapfeltételezés attól függ, hogy milyen utasításban használjuk az előre-hivatkozást. Vezérlésátadó utasításban az alapfeltételezés NEAR típusú címke. Egyéb utasításokban a fordító olyan változót feltételez, amely a DS szegmensregiszterhez relatív. Abban az esetben, ha az alapfeltételezés szerinti típus és szegmensregiszter nem felel meg az azonosító programbeli használatának, fordítási hiba keletkezik.

Az ilyen hibákat a címkiszámítás viszonyítási alapjának megváltoztatásával (:), valamint a PTR és SHORT művelet használatával kerülhetjük el. Ha egy olyan változóra hivatkozunk előre, amely az ES, SS vagy CS regiszterhez relatív, akkor a címkiszámítás viszonyítási alapját megváltoztató művelettel (:) kell meghatározni azt a szegmensregisztert, szegmenset vagy csoportot, amelyhez a változó relatív.

Például a

```
MOV AX,SS:azonosító
```

hatására az AX regiszterbe az SS szegmensregiszterhez relatív azonosítóval jelölt változó tartalma kerül.

A címkiszámítás viszonyítási alapjának megváltoztatása nélkül a fordító azt feltételezi, hogy a változó a DS regiszterhez relatív.

Ha egy JMP (feltétel nélküli ugrás) utasításban olyan utasítás címkéjére hivatkozunk előre, amely a hivatkozás helyétől 128 byte-nál kisebb távolságra van, akkor célszerű a SHORT műveletet használni. A SHORT művelet elhagyása nem okoz ugyan hibát, de ekkor a fordító azt feltételezi, hogy az utasítás, amelynek a címkéjére előre-hivatkoztunk,

128 byte-nál nagyobb távolságra van. Ilyenkor a fordító egy NOP utasítást is generál, amely 128 byte-nál kisebb távolság esetén felesleges.

Példa:

```
JMP SHORT címke
```

A SHORT művelet azt jelzi, hogy a későbbiekben deklarált "címke" a JMP utasítás helyétől 128 byte-nál kisebb távolságra van.

Ha CALL vagy JMP utasításban előre hivatkozunk egy utasítás címkéjére, akkor a PTR művelettel meghatározhatjuk a címke típusát (NEAR vagy FAR). A fordító alapfeltételezése a NEAR típus, így ezt a típust nem kötelező PTR művelettel megadni. Ha a címke FAR típusú és nem használjuk a PTR műveletet, akkor fázishiba keletkezik. A fázishiba azt jelenti, hogy az adott címke típusa nem azonos a fordítás első és második menetében (NEAR típusúként használtuk a későbbiekben FAR típusúként deklarált címkét).

Példa a címke típusának meghatározására:

```
CALL FAR PTR számol
```

A PTR művelet hatására a fordító FAR típusúnak tekinti a későbbiekben deklarált "számol" címkét. Ha egy szegmensnévre úgy hivatkozunk előre, hogy a programban először egy címkiszámítás viszonyítási alapjaként szerepel, akkor előbb a GROUP direktívával rendeljük hozzá a szegmensnevet egy csoportnévhez, majd az ASSUME direktívával rendeljük egy szegmensregiszterhez a csoportnevet! Ha ezt nem tesszük meg, akkor a fordító ignorálhatja a címkiszámítás viszonyítási alapját megváltoztató műveletet és az alapfeltételezés szerinti szegmensregisztert használja, ami általában hibát okoz a fordítás második menetében.

Példa a helyes használatra:

```
csoport     GROUP verem
            ASSUME SS:csoport
prog        SEGMENT
            .
            .
            .
            ADD AX, verem:elem
            .
            .
            .
```

5.2.7 Globális deklarációk

A lokális címkékre, változókra és egyéb szimbólumokra csak abban a modulban lehet hivatkozni, amelyikben a deklarációjuk szerepel. A globális deklarációk direktívái lehetővé teszik a programozó számára, hogy olyan cimkéket, változókat és egyéb

szimbólumokat definiáljon, amelyek a program bármely moduljából elérhetők. Az ilyen szimbólumokat **globális szimbólumoknak** nevezzük. A globális deklarációk direktívái:

PUBLIC
EXTRN

A PUBLIC direktíva hatására az adott modulban lokálisan definiált szimbólumok globálisakká válnak, azaz elérhetőek lesznek a program bármely más moduljából is. A globális szimbólumot a program pontosan egy moduljában kell deklarálni, és e modulban adjuk meg azt is a PUBLIC direktívával, hogy a szimbólum globális. Ezután egy másik modulban a szimbólumot úgy tesszük ismertté, hogy itt viszont külső (external) szimbólumként adjuk meg ay EXTRN direktívával. A globális szimbólum tetszőleges számú modul az EXTRN direktívájában előfordulhat.

A PUBLIC direktíva alakja:

PUBLIC *azonosító*[,,,]

Az "*azonosító*" egy olyan változó, címke vagy abszolút szimbólum azonosítója, amelyet a PUBLIC direktívát tartalmazó modulban deklaráltunk. A PUBLIC direktívában megadott szimbólum csak olyan lehet, amelynek értéke 1 vagy 2 byte-os egész szám vagy string.

Az azonosítóban szereplő kisbetűket a fordító nagybetűkké konvertálja mielőtt az azonosítót az object file-ba másolja. Ha az object file-ban is meg akarjuk őrizni a kisbetűket, akkor a fordító hívásakor meg kell adnia /ML és /MX opciót. Erről az V. fejezet 3. pontjában található bővebb információ.

Az EXTRN direktíva alakja:

EXTRN *azonosító:típus*[,,,]

Az "azonosító" egy olyan változó, címke vagy szimbólum azonosítója, amelyet a program egy másik moduljában deklaráltunk, és ott PUBLIC direktívában is szerepel. Az EXTRN direktívában megadott típusnak meg kell egyeznie azzal a típussal, amelyet abban a modulban adtunk meg, ahol a szimbólumot deklaráltuk. A "*típus*" a következő kulcsszavak valamelyikével adhatók meg:

BYTE	WORD
DWORD	QWORD
TBYTE	
NEAR	FAR
ABS	

Az ABS típus azt jelenti, hogy a szimbólum értéke egy konkrét szám.

Mivel az aktuális cím csak a program összeszerkesztésekor alakul ki, minden, az EXTRN direktívával megadott, azaz külső szimbólumról a fordító feltételezi, hogy a program valamely moduljában PUBLIC-ként is deklaráltuk. A fordító egy alapfeltételezést állít fel arról, hogy ez a szimbólum a program melyik szegmensében helyezkedik el.

Ez az alapfeltételezés attól függ, hogy az EXTRN direktíva hol helyezkedik el a modulban. Ha a direktíva egy szegmens belsejében található, akkor a fordító azt feltételezi, hogy a külsőként megadott azonosító az EXTRN direktívát tartalmazó szegmens szegmensregiszteréhez relatív. Ebben az esetben a szimbólum EXTRN és PUBLIC deklarációjának azonos nevű és azonos attributumokkal rendelkező szegmensekben kell szerepelnie. Ha a direktíva helye minden szegmensen kívülre esik, akkor nincs alapfeltételezés. Ekkor az elem PUBLIC deklarációja a program bármely moduljának bármely szegmensében előfordulhat.

A külső elem alapfeltételezés szerinti szegmensének megváltoztatására a címkiszámítás viszonyítási alapját megváltoztató műveletet (:) használhatjuk.

5.3 Makrok definiálása és hívása

A makró-definíciót úgynevezett makró-direktívák és assembly utasítások segítségével építjük fel. A macro-assembler a következő makró-direktívák használatát engedi meg:

MACRO
LOCAL
PURGE
REPT
IRP
IRPC
EXITM
ENDM

A MACRO direktívával a programozó egy adott nevű makrót definiál, amely legalább egy forrás-nyelvű utasításból áll. A programban, azaz a forrás-file-ban a fordító a makró minden előfordulását, azaz a makró hívásait automatikusan helyettesíti a makró-definícióban megadott utasítás-sorozattal.

A LOCAL direktíva lehetővé teszi, hogy a makróban egyedi címkéket definiáljunk. A PURGE direktíva a makró-definíció törlésére használható. A REPT, IRP és IRPC direktívák segítségével adott utasítás-sorozatok folyamatosan ismételhetők. Az ismétlési tényezőt egy kifejezés értéke, egy paraméterlista elemeinek száma vagy egy sztringben lévő karakterek száma határozza meg.

A makró-direktívában speciális műveleti jeleket lehet használni, amelyek a következők:

&	a helyettesítés jele
;;	a makró-megjegyzés jele
!	a karakter-konstans jele
< >	a szöveg-konstans jele
%	a kifejezés jele

Ezeknek a műveleti jeleknek a makró-definícióban való használatával speciális vezérlési műveleteket lehet végrehajtani.

5.3.1 A MACRO és az ENDM direktíva

A MACRO direktíva a makró-definíció kezdetét, az ENDM direktíva a makró-definíció végét jelzi. A makró-definíciók általános alakja:

makrónév **MACRO** [*formális paraméter*[„„]]
utasítás
.
.
.
ENDM

A *makrónév*nek egyedinek kell lennie. Ezt a nevet használjuk a forrás-file-ban a makró hívására. A formális paraméterek olyan azonosítók, amelyek a makró hívásakor kapnak értéket. Tetszőleges számú formális paraméter adható meg, de el kell férniük a MACRO direktíva sorában.

A makró törzse tetszőleges számú makró-assembly utasításból állhat, amelyekben a formális paraméterek akárhányszor előfordulhatnak. A makró-törzsben MACRO direktíva megadása is megengedett, tehát a makró-definíciók egymásba ágyazhatók.

A makró hívása annyiszor történik meg, ahányszor a makrónévre hivatkozunk a modulban. A megjegyzésekben előforduló makróneveket a fordító nem veszi figyelembe. A makróhívás helyére bemásolja a makró törzsében megadott utasításokat, úgy, hogy a formális paramétereket a híváskor megadott értékekkel, az úgynevezett aktuális paraméterekkel helyettesíti. Az aktuális paramétereket a makró-definícióban felírt formális paraméter sorrend szerint kell megadni: pl. a makró-definícióban másodikként megadott formális paraméter a makró-hívás második aktuális paraméterével helyettesítődik.

A makró-definícióba ágyazott másik makró-definíciót addig nem dolgozza fel a fordítóprogram, amíg a külső makró hívása meg nem történik. Ez azt jelenti, hogy egy makróba ágyazott makrót addig nem lehet hívni, ameddig a programban legalább egyszer nem hívtuk a külső makrót. A makró-definíciók tetszőleges mélységben egymásba skatulyázhatók. A mélységet csak a modul fordításakor rendelkezésre álló memóriaterület mérete korlátozza. A makró-definíció más makrók hívását is tartalmazhatja. *A makrók nem lehetnek rekurzívek, azaz törzsükben nem hívhatók mégegyszer.* Makrókifejtéskor a fordítóprogram a formális paramétereket mindig helyettesíti a makróhívásban megadott aktuális paraméterekkel. Ezért nem célszerű formális paraméterként például regiszternevet használni, mert a makró-definícióban lévő olyan utasítások, amelyek erre a regiszterre hivatkoznak, a makró kifejtésekor helytelenül kerülnének a programba.

A makró utasításait csak a makró hívásakor fordítja le a makró-assembler és csak azon a helyen, ahol az utasításokat a forrás-file-ba illeszti. Ebből következik, hogy a lefordított kód címei a hívás helyéhez, és nem a definíció helyéhez relatívek. *Maga a makró-definíció sohasem fordítódik le.*

Abban az esetben, ha a MACRO direktívát közvetlenül a TITLE, SUBTTL vagy NAME direktíva után használjuk, a fordító TITLE, SUBTTL vagy NAME nevű makrót kezd generálni, így megváltozhat ezen direktívák eredeti hatása. (A TITLE direktívával program listázásakor kiírandó cím szövegét, a SUBTTL direktívával az alcímet, a NAME direktívával a szerkesztő hibajelzéseiben kiírandó modulnevet lehet meghatározni.) Ha ezt a problémát el akarjuk kerülni, akkor a címben, alcímben vagy modulnévben szereplő MACRO szót meg kell változtatni, például kiegészíthetjük egy speciális karakterrel: "-MACRO". (A TITLE, SUBTTL és NAME direktívával kapcsolatban e fejezet 4. pontjában található bővebb információ.)

Makródefinícióval más direktívák és assembly utasítások működése is megváltoztatható.

Példa makró definiálására:

```
kivon MACRO a,b,c
      MOV  AX,a
```

```
SUB  AX,b
MOV  c,AX
ENDM
```

5.3.2 Makróhívások

A már definiált makrók hívása a következő formában történik:

makrónév [*aktuális_paraméter*[„„]]

A makróhívás hatására a fordító bemásolja a hívás helyére a makrónévvel azonosított makró definíciójában szereplő utasításokat és az utasításokban lévő formális paramétereket a hívásban megadott aktuális paraméterekkel helyettesíti. Aktuális paraméterként azonosítót, konstanst vagy értéklistát lehet megadni. Az aktuális paraméterek száma tetszőleges lehet, de nem folytathatók új sorban.

A fordító az első formális paramétert az első aktuális paraméterrel helyettesíti, a másodikat a másodikkal és így tovább. *Ha a makróhívásban több aktuális paramétert adtunk meg, mint a makró-definícióban lévő formális paraméterek száma, akkor a fordító a felesleges aktuális paramétereket nem veszi figyelembe. Ha a makróhívásban megadott aktuális paraméterek száma kevesebb, mint a definícióban lévő formális paraméterek száma, akkor a makró kifejtése során azok a formális paraméterek, amelyeknek nem feleltethetők meg aktuális paraméterek, kimaradnak az utasításokból, semmivel sem helyettesítődnek.*

Abban az esetben, **ha egy aktuális paraméterként egy értéklistát akarunk megadni, a listát csúcsos zárójelek (< >) közé kell tenni**. A lista elemeit vesszővel kell elválasztani egymástól.

Értéklista aktuális paraméterként való megadásának akkor van értelme, ha a hívott makró törzsében olyan IRP ismétlő direktíva is szerepel, amelynek helyettesítő paramétere azonos az őt tartalmazó makródefiníció valamelyik formális paraméterével.

(Az IRP olyan ismétlő direktíva, amelyben az ismétlések számát a helyettesítő paraméterek száma határozza meg. A direktíva részletes leírása e fejezet 3.10 pontjában található.)

Példa makróhívásra:

```
kivon 10,csökk,marad
```

Három aktuális paraméterrel hívja a "kivon" makrót, amelynek az előző pontban megadott definíciója szerint a következő utasítás-sorozat generálódik:

```
MOV AX,10
SUB AX,csökk
MOV marad,AX
```

5.3.3 Részkaraktersorozat megadása makró-paraméterrel

Valamely karaktersorozat egy részét úgy is megadhatjuk, hogy a helyettesítés jelét (&) és a makró valamely formális paraméterét használjuk. A helyettesítés jelét bárhol lehet alkalmazni, ahol a formális paraméter közvetlenül egy másik karakter előtt vagy után áll,

vagy ha a formális paraméter egy olyan karaktersorozatban helyezkedik el, amely aposztrófok között áll.

A helyettesítés jele: &

Megadása:

karaktersorozat&formális_paraméter

vagy

formális_paraméter&karaktersorozat

Példa a helyettesítés jelének használatára:

```
peld  MACRO x
abx   MOV  bx,'&x'
ab&x  JMP  cimke
      ENDM
```

Ha az előzőleg definiált makrót a következőképpen hívjuk:

```
peld a
```

akkor a fordító a következő két utasítást generálja:

```
abx   MOV bx,'a'
aba   JMP cimke
```

Egymásba ágyazott makrók esetében a formális paraméter szimbólumban vagy karaktersorozat-konstansban előforduló, aktuális paraméterrel való helyettesítésekor annyi "&" jelet kell használni, amennyi az ágyazási mélység.

Tekintsük például a következő makró-definíciót:

```
mezők MACRO p
      IRP   q, <1,2,3>  ;;q az IRP direktíva
p&&q  DB    q           ;; formális paramétere
      ENDM
      ENDM
```

(Az első & a "p"-hez, a második a "q"-hoz tartozik.)

A makró hívásakor a "p" formális paraméter azonnal helyettesítődik. A "q" formális paraméter azonban csak akkor helyettesítődik, amikor a fordító feldolgozza az IRP direktívát. (Az IRP direktíva részletes ismertetése e fejezet 3.10 pontjában található. Hatása: egy utasítás-sorozatot annyiszor ismétel, amennyi az IRP direktívában megadott helyettesítő paraméterek száma.) A "q" formális paraméter az IRP paraméterlistájában megadott értékek szerint helyettesítődik. Ha a makró hívása például:

```
mezők mez
```

akkor a fordítóprogram a következő sorokat generálja:

```
mez1  DB  1
mez2  DB  2
mez3  DB  3
```

5.3.4 A szöveg-konstans jele

A szöveg-konstans jelének hatására a fordító-program literálnak tekinti a szöveget. A szöveg-konstans megadási módja:

<szöveg>

A szöveg-konstans leggyakrabban makró-hívásokban és az IRP direktívában használjuk abból a célból, hogy a fordító a paraméterlista elemeit egyetlen paraméternek tekintse.

A szöveg-konstans jelét arra is felhasználhatjuk, hogy a fordító literálként értelmezzen speciális karaktereket. Például

<;>

nem azt jelenti, hogy megjegyzés következik, hanem a ";" karaktert.

A szöveg-konstans legfeljebb a sor végéig tarthat. A fordítóprogram mindig eltávolítja a csúcsos zárójelpárt, valahányszor a paraméter előfordul a makróban. Ha egymásba ágyazott makrókat használunk, akkor annyi csúcsos zárójelpárt kell megadni, amennyi az ágyazási mélység.

5.3.5 A karakter-konstans jele

A karakter-konstans megadása:

!karakter

Ennek hatására a fordítóprogram literálként értelmezi a karaktert. Akárcsak a szöveg-konstans jelét, a karakter-konstans jelét is használhatjuk arra, hogy a fordító a speciális karaktereket literálnak tekintse. Például

!; és <;>

azonos hatású.

5.3.6 A kifejezés jele

A kifejezés megadási módja:

%szöveg

A kifejezés jelének hatására a makró-assembler a szöveget kifejezésnek tekinti. Az aktuális számrendszert használva kiszámítja a kifejezés értékét és a szöveget ezzel az értékkel helyettesíti. A "szöveg" helyes assembly nyelvű kifejezés kell, hogy legyen.

A kifejezés jelét olyan makró-hívásokban használjuk, amelyekben nem a kifejezés szövegét, hanem a kifejezés kiszámított értékét akarjuk átadni a makrónak.

A kifejezés legfeljebb a sor végéig tarthat.

Definiáljunk egy olyan makrót, amely képernyőre ír egy szöveget és egy értéket!

```
kiír   MACRO szöveg, érték
       %OUT  * szöveg, érték *
       ENDM
```

Ha például

```
e1 EQU 10
e2 EQU 5
```

akkor a

```
kiír <e1-e2= >,%(e1-e2)
```

makróhívás hatására a "szöveg" formális paraméternek az "e1-e2=" szöveg-konstans aktuális paraméter felel meg, az "érték" formális paraméter helyére pedig az 5 numerikus érték helyettesítődik be.

5.3.7 A PURGE direktíva - makródefiníciók törlése

A PURGE direktívával adott nevű makrók definícióját törölhetjük a memóriából. A törölt makrók további hívása hibajelzést okoz.

A PURGE direktíva alakja:

PURGE *makrónév*[,,,]

Legfeljebb annyi makrónév adható meg, amennyi a direktíva sorában elfér.

A direktíva csak akkora területet töröl, amekkora helyet a makró-definíció elfoglalt. Ha "makrónév" megegyezik valamely utasítás vagy direktíva mnemonikus kódjával, akkor a PURGE direktíva visszaállítja a mnemonikus kód eredeti jelentését.

A makró-könyvtár egy olyan file, amely makró-definíciókat tartalmaz. A makró-könyvtárat az INCLUDE direktívával lehet a forrás-file-ba illeszteni, és ezután a PURGE direktívával törölhetjük a felesleges makró-definíciókat.

Ha egy makrót újra definiálunk, akkor a makró előző definíciója automatikusan törlődik.

Példa:

```
PURGE makro1,makro2
```

5.3.8 A LOCAL direktíva - egyedi azonosítók lérehozása makrókban

A LOCAL direktíva hatására a makró egyes kifejtéseire nézve egyedi azonosítók generálódnak.

Alakja:

LOCAL *dummy-azonosító*[,,,]

A "*dummy-azonosító*" a makró minden újabb kifejtésekor új egyedi névvel helyettesítődik. A "*dummy-azonosító*" a makró bármely utasításában előfordulhat. A kifejtéskor

??xxxx

alakú egyedi nevet helyettesít be a fordító, ahol xxxx egy [0000,FFFF] intervallumbeli hexadecimális szám.

A LOCAL direktívát rendszerint arra használjuk, hogy a makrókban egyedi címkéket állítsunk elő. Ha ugyanis nem adunk meg LOCAL direktívát, és a makrót többször hívjuk, a kifejtés eredményeképpen többszörösen definiált címkéket kapunk, amelyekre a makró-assembler hibajelzést ad. Ezt a problémát úgy lehet elkerülni, hogy a makró-definícióban szereplő összes címkét LOCAL direktívával adjuk meg, így a makró minden egyes kifejtésekor új, egyedi címkék generálódnak.

A LOCAL direktívát kizárólag makró-definícióban lehet használni és a definíció összes többi utasítását meg kell előznie.

Példa:

```
mac   MACRO a,b
      LOCAL címke
      MOV  AX,b
címke MOV  CX,a
      INC  AX
      JNZ  címke
      ENDM
```

5.3.9 A REPT és az ENDM direktíva - utasításcsoport ismétlése

A REPT, ENDM direktívapárral egy ismétlendő utasításcsoportot lehet definiálni a következőképpen:

REPT *kifejezés*
utasítások
ENDM

Az ismétlések számát a *"kifejezés"* értéke határozza meg. A kifejezés értéke 16 biten ábrázolható, előjel nélküli szám kell, hogy legyen. Az ismétlési tényező az aktuális alapértelmezés szerinti számrendszerben értendő. Az utasítások tetszőleges, helyesen megadott makró-assembly utasítások lehetnek.

Példa:

```
a     =    0
      REPT5
a     =    a+2
      DW   a
      ENDM
```

A REPT direktíva hatására az = direktíva és a DB direktíva 5-ször ismétlődik. Ennek hatására 5 szónyi adat generálódik a 2,4,6,8,10 értékekkel.

5.3.10 Az IRP és az ENDM direktíva - utasításcsoport ismétlése paraméterhelyettesítéssel

Az IRP, ENDM direktívapárral egy ismétlendő utasításcsoportot definiálhatunk. Az ismétlések számát az IRP direktívában megadott paraméterlista elemeinek száma határozza meg. Az ismétlendő utasításcsoport definiálása a következőképpen történik:

IRP *helyettesítendő paraméter*, <*helyettesítő paraméter*,,,>
utasítások
ENDM

Az utasításcsoport ismétléseinek számát a csúcsos zárójelek között megadott "*helyettesítő paraméter*"-er száma határozza meg. A "*helyettesítendő paraméter*"-rel azt az objektumot jelöljük ki, amelyet az IRP és az ENDM közötti utasításokban való minden előfordulásakor helyettesíteni akarunk az aktuális helyettesítő paraméterrel. A "*helyettesítő paraméter*" tetszőleges, helyesen megadott szimbólum, sztring, numerikus vagy karakter-konstans lehet. Tetszőleges számú helyettesítő paraméter megadható. Több helyettesítő paraméter megadása esetén a paramétereket veszszővel kell elválasztani egymástól. Az "*utasítások*" tetszőleges makró-assembly utasítások lehetnek, amelyekben a helyettesítendő paraméter akárhányszor előfordulhat.

Amikor a fordító egy IRP direktívával találkozik a forrásprogramban, akkor annyiszor másolja be az IRP és az ENDM közötti utasítássorozatot, ahány helyettesítő paramétert megadtunk. Az utasítássorozat első másolatában a helyettesítendő paraméter minden előfordulását az elsőként megadott helyettesítő paraméterrel írja felül, a második másolatban a másodikként megadott helyettesítő paraméterrel, és így tovább. Ha a helyettesítő paraméterek listájában üres paraméter is van, akkor a megfelelő másolatban a helyettesítendő paraméter helye üresen marad. Abban az esetben, ha a helyettesítendő paraméterek listája üres, a fordító nem veszi figyelembe az IRP direktivát és egyetlen utasítást sem másol be.

Az IRP direktíva makró-definíció belsejében is előfordulhat. Ha az itt megadott IRP direktíva helyettesítő paramétere megegyezik a makró formális paraméterével, akkor ezt a formális paramétert kell az IRP direktívában csúcsos zárójelek között megadni. Tekintsük például a következő makró-definíciót:

```
adat   MACRO a
       IRP  d,<a>
       DW   d
       ENDM
       ENDM
```

Ha a makrót a következőképpen hívjuk:

```
adat  <1,2,3>
```

akkor a makró kifejtése:

```
IRP d,<1,2,3>
DW d
ENDM
```

Ebből látható, hogy a makró kifejtésekor a hívásban megadott csúcsos zárójeleket elhagyja a fordító.

Az IRP direktívát természetesen nemcsak makró-definíció belsejében használhatjuk. Az

```
IRP d,<1,2,3>
DW d
ENDM
```

hatására a fordító a következő utasítássorozatot generálja:

```
DW 1
DW 2
DW 3
```

5.3.11 Az IRPC és az ENDM direktíva

Az IRPC, ENDM direktívapárral egy ismétlendő utasításcsoportot definiálhatunk. Az ismétlések számát az IRPC direktívában megadott karaktersorozat karaktereinek száma határozza meg. Az ismétlendő utasításcsoport definiálása a következőképpen történik:

IRPC *helyettesítendő paraméter,karaktersorozat*
utasítások
ENDM

Az utasításcsoport ismétléseinek számát a sztring karaktereinek száma határozza meg. A *"helyettesítő paraméter"*-rel azt az objektumot jelöljük ki, amelyet az IRPC és az ENDM közötti utasításokban való minden előfordulásakor helyettesíteni akarunk a string aktuális karakterével. A *"karaktersorozat"*-ban betűk, számjegyek és egyéb karakterek lehetnek. Ha a karaktersorozatban szóköz, vessző vagy egyéb elválasztó karakter is szerepel, akkor a sztringet csúcsos zárójelek között kell megadni. Az *"utasítások"* tetszőleges makró-assembly utasítások lehetnek, amelyekben a helyettesítendő paraméter akárhányszor előfordulhat.

Amikor a fordító egy IRPC direktívát talál a forrásprogramban, akkor annyiszor másolja be az IRPC és az ENDM között álló utasítássorozatot, amennyi az IRPC direktívában megadott string karaktereinek száma. Az utasítássorozat első másolatában a helyettesítendő paraméter minden előfordulását a string első karakterével írja felül, a második másolatban a string második karakterével, és így tovább.

Példa:

```
IRPC a,123
DW a
ENDM
```

hatására a fordító 3 db DW direktívát generál:

```
DW 1
DW 2
DW 3
```

5.3.12 Az EXITM direktíva

Az EXITM direktívát makrókifejtésből vagy egy utasítássorozat ismétlődő kifejtésekből való kilépésre használjuk.

Alakja:

EXITM

A kilépés után a fordítás a makróhívás vagy ismétlési sorozat utáni első utasítással folytatódik. Az EXITM direktívát általában a feltételes (IF) direktívákkal együtt arra használjuk, hogy a makrókifejtést vagy az utasítássorozat ismétlését vezéreljük.

Ha a fordító egy EXITM direktívát talál, akkor azonnal abbahagyja a makró kifejtését vagy az ismétlődő utasítássorozatok generálását, és a további utasításokat nem dolgozza fel. Ha az EXITM olyan makróban vagy ismétlési blokkban szerepel, amelyet egy másik makróban vagy ismétlési blokkban definiáltunk, a kifejtés a tartalmazó makróban illetve ismétlési csoportban folytatódik.

A feltételes direktívák ismertetése e fejezet következő pontjában található.

Példa:

```
makróMACRO param
        utasítás
        .
        .
        .
        IF kifejezés
          EXITM
        ELSE
          utasítás
        ENDIF
        ENDM
```

Az itt definiált makró hívásakor a fordító feldolgozza az IF direktíva előtti utasításokat, majd kiértékeli az IF direktívában megadott kifejezést. Ha a kifejezés értéke nulla, akkor az EXITM direktívát dolgozza fel, és abbahagyja a makró kifejtését.

5.3.13 Makró-megjegyzések

A makró-megjegyzések olyan magyarázó szövegek, amelyek a makródefinícióban állnak, de a makró kifejtésekor nem kerülnek be a programba.

Alakja:

;;megjegyzés

A két pontosvessző utáni szöveget a fordító nem veszi figyelembe. Ez a szöveg csak a forrás-listában, a makró definíciójában jelenik meg.

A makró-assembly program egy pontosvesszővel kezdődő megjegyzései, a makró-megjegyzésektől eltérően, bekerülhetnek a makró minden kifejtésébe, ha megadjuk a .XALL direktívát (ld. ebben a fejezetben a 4.6 pontot).

5.3.14 Feltételes fordítás

A makró-assembly nyelvben feltételes direktívákat is lehet használni, amelyekkel feltételes fordítást valósíthatunk meg. A feltételes fordítás azt jelenti, hogy egy utasítássorozatot csak akkor fordítunk le, ha egy adott feltétel teljesül. A feltételes direktívák a következők:

IF
IFE
IF1
IF2
IFDEF
IFNDEF
IFB
IFNB
IFIDN
IFDIF
ELSE
ENDIF

Az IF, ELSE és ENDIF direktívák határolják azokat az utasításokat, amelyeket feltételesen akarunk fordítani. A feltételes blokkokat a következőképpen kell megadni:

IF [*feltétel*]
utasítások
[**ELSE**
utasítások]
ENDIF

ahol az utasítások tetszőleges macro-assembly utasítások lehetnek, beleértve újabb IF direktívákat is. Az ELSE direktíva használata nem kötelező.

A macro-assembler csak akkor fordítja le az IF direktíva után lévő utasításokat, ha az adott IF direktívának megfelelő feltétel teljesül. Ha ELSE direktívát is megadtunk, akkor a feltétel teljesülése esetén a blokk utolsó lefordítandó utasítása az az utasítás, amely közvetlenül az ELSE előtt áll. Az ELSE direktívát követő utasítások fordítása csak akkor következik be, ha az IF direktívához tartozó feltétel nem teljesül. A feltételes blokk végét az ENDIF direktíva jelzi. Egy IF direktívához legfeljebb egy ELSE direktíva használata megengedett.

Az IF direktívák legfeljebb 255 szint mélységig ágyazhatók egymásba. Az egyértelműség biztosítása miatt a fordító minden ELSE direktívát a hozzá legközelebbi előző IF direktívához tartozónak tekint.

5.3.14.1 Az IF és az IFE direktíva

Az IF és az IFE direktíva alakja:

IF *kifejezés*
IFE *kifejezés*

Az IF és az IFE direktíva kiértékeli a "*kifejezés*"-t. Az IF direktíva szerint akkor teljesül a feltétel, ha a kifejezés értéke nem nulla. Az IFE direktíva akkor találja igaznak a feltételt, ha a kifejezés értéke nulla.

Az IF és az IFE direktívában olyan kifejezést kell megadni, amelynek konkrét értéke kiszámítható és nem tartalmaz postdefinit hivatkozást.

Példa:

```
IF v-AH
v = v + 1
ENDIF
```

5.3.14.2 Az IF1 és IF2 direktíva

Az IF1 és az IF2 direktíva alakja:

IF1
IF2

Az IF1 és az IF2 direktíva az aktuális fordítási menetet vizsgálja. Az IF1 direktíva hatására a megfelelő utasítássorozat csak az első menetben fordítódik le, az IF2 hatására csak a második menetben. Az IF1 és IF2 direktívának nincs argumentuma.

5.3.14.3 Az IFDEF és az IFNDEF direktíva

Az IFDEF és az IFNDEF direktíva azt vizsgálja, hogy a megadott "azonosító" egy előzőleg már definiált címke, változó vagy szimbólum. Az IFNDEF direktíva szerint akkor teljesül az adott utasítássorozat fordításához szükséges feltétel, ha az "azonosító" még nem definiált.

Tetszőleges "*azonosító*" megadható. Ha az azonosító előre való hivatkozás, akkor az első fordítási menetben definiálatlannak, a másodikban definiáltnak minősül. Ez a fázishibák gyakori oka.

Példa:

```
IFNDEF változó
változó DQ ?
ENDIF
```

5.3.14.4 Az IFB és az IFNB direktíva

Az IFB és az IFNB direktíva alakja:

IFB <*argumentum*>
IFNB <*argumentum*>

Az IFB és az IFNB direktíva az adott "*argumentum*"-ot vizsgálja. Az IFB direktíva hatására akkor történik meg az utasítássorozat fordítása, ha az argumentum üres. Az IFNB direktíva akkor tekinti a feltételt igaznak, ha az argumentum nem üres.

Az "argumentum" tetszőleges azonosító, szám, vagy kifejezés lehet. Kötelező csúcsos zárójelek közé tenni.

Az IFB és IFNB direktívákat főként makró-definíciókban célszerű használni. Az IFB és IFNB segítségével a makróhívásban átadott paraméterek szerint vezérelhetjük a makró utasításainak feltételes kifejtését. Ekkor az "argumentum" a MACRO direktívában megadott egyik formális paraméter.

Példa:

IFB <fp>

Az IFB direktíva a "fp" argumentumot vizsgálja, amely egy makró-definícióban formális paraméter. Ha a makró hívásakor nem adtunk ehhez a formális paraméterhez aktuális paramétert, akkor a megfelelő utasítássorozat lefordítódik.

5.3.14.5 Az IFIDN és az IFDIF direktíva

Az IFIDN és az IFDIF direktíva alakja:

IFIDN <*argumentum1*>,<*argumentum2*>
IFDIF <*argumentum1*>,<*argumentum2*>

Az IFIDN és az IFDIF direktíva az "*argumentum1*"-et összehasonlítja az "*argumentum2*"-vel. Az IFIDN direktíva hatására akkor fordítódik le a megfelelő utasítássorozat, ha a két argumentum azonos, az IFDIF direktíva hatására akkor, ha a két argumentum **különböző**.

Az argumentumok tetszőleges azonosítók, számok vagy kifejezések lehetnek. A két argumentum akkor azonos, ha karakterről-karakterre megegyeznek. **A csúcsos zárójelek kiírása kötelező.**

Az IFIDN és az IFDIF direktívákat főként makró-definíciókban célszerű használni. Az IFIDN és az IFDIF segítségével a makróhívásban átadott paraméterek segítségével vezérelhetjük a makró utasításainak feltételes kifejtését. Az ilyen esetekben megadott argumentumoknak szerepelniük kell a MACRO direktíva formális paraméter-listájában.

Példa:

IFIDN <a>,<b>

Az IFIDN direktíva összehasonlítja az "a" és "b" argumentumot. Ha ezek egy makródefiníció formális paraméterei, és makróhíváskor "a"-hoz és "b"-hez azonos aktuális paramétert rendeltünk, akkor a megfelelő utasítássorozat lefordítódik.

5.4 A macro-assembler file-kezelésének vezérlése

A makró-assembler lehetőséget nyújt olyan direktívák használatára is, amelyek a fordítás során a makró-assembler által olvasandó és előállítandó forrás-, object- és lista-file-ok kezelését vezérlik. A következő direktívák adhatók meg a file-kezelés vezérlésére:

INCLUDE	Forrás-file bemásolása
.RADIX	A forrás-file-ban lévő számok alapértelmezés szerinti számrendszerének beállítása
%OUT	Üzenetek kiírása a konzolra
NAME	Modulnév beírása az object-file-ba
TITLE	A programlista címének megadása
SUBTTL	A programlista alcímének megadása
PAGE	A programlista lapméretének beállítása, feltétel nélküli lapdobás előírása
.LIST	Utasítások kiírása a programlistába
.XLIST	Utasítások elhagyása a programlistából
.LFCOND	Utasítások listázása, ha a feltétel nem teljesül az IF direktívákban
.SFCOND	Utasítások elhagyása a programlistából, ha a feltétel nem teljesül az IF direktívákban
TFCOND	Feltételes blokkok listázási módja alapértelmezésének beállítása
.LALL	Makrókifejtések listázása
.SALL	Makrókifejtések elhagyása a programlistából
.XALL	Megjegyzések kihagyása a makrók listájából
.CREF	Szimbólumok beírása a keresztreferencia-file-ba
.XCREF	Szimbólumok kihagyása a keresztreferencia-file-ból

5.4.1 Az INCLUDE direktíva

Az INCLUDE direktíva segítségével egy adott forrás-file-t másolhatunk be fordítás közben az aktuális forrás-file-ba. Az INCLUDE direktíva alakja:

INCLUDE *file-név*

A "file-név" egy létező file neve. Ha a file nem abban a directory-ban található, amely a fordításkor aktuális, akkor meg kell adni a file-t tartalmazó directory-t. Ha a makró-assembler nem találja az adott file-t, akkor hibajelzést ad és a fordítás megszakad.

Amikor a fordító egy INCLUDE direktívát talál, megnyitja az adott file-t és azonnal elkezdi fordítani a megnyitott file forrás-utasításait. Miután a file minden utasításának a fordítása megtörtént, a makró-assembler az INCLUDE direktívát követő utasítás feldolgozásával folytatja tevékenységét.

Az INCLUDE direktívák egymásba ágyazhatók, ami azt jelenti, hogy a direktívában megadott file is tartalmazhat INCLUDE direktívákat. A programlistában a macro-assembler "c" betűvel jelzi az INCLUDE direktíva hatására beillesztett utasításokat.

Példák:

INCLUDE struktúrák
INCLUDE \dir\struktúrák
INCLUDE C:\dir\struktúrák

5.4.2 A TITLE direktíva

A TITLE direktívát a programlista címének meghatározására használhatjuk a következőképpen:

TITLE *szöveg*

A direktíva hatására a makró-assembler kiírja a "szöveg"-et a forráslista minden új lapjának első sorába. A "szöveg" tetszőleges karakterekből állhat és legfeljebb 60 karakter hosszúságú lehet. Egy modulban legfeljebb egy TITLE direktíva adható meg.

Ha a modul nem tartalmaz NAME direktívát, akkor a fordító a TITLE direktívában megadott "szöveg" első 6, szóköztől különböző karakterét tekinti modulnévnek.

Példa:

TITLE 1. program

5.4.3 A SUBTTL direktíva

A SUBTTL direktíva segítségével a programlista alcímét határozhatjuk meg. Alakja:

SUBTTL *szöveg*

A SUBTTL direktíva hatására a makró-assembler a TITLE direktívával a programlistába írt *szöveg*et követő sorba kiírja a SUBTTL direktívával megadott alcímet. Az alcím a programlista minden új lapján megjelenik. A szöveg tetszőleges karakterekből állhat. A fordító csak az alcím első 60 karakterét veszi figyelembe. Ha a SUBTTL direktívával nem adunk meg szöveget, akkor az alcím sora üresen marad. Egy programban tetszőleges számú SUBTTL direktíva megadható. Minden újabb SUBTTL direktíva új szöveggel helyettesíti az előző alcímet.

Példák:

SUBTTL ellenőrzések
SUBTTL

5.4.4 A PAGE direktíva

A PAGE direktívával beállíthatjuk a programlista lapméretét, növelhetjük a szakasz-számot vagy lapozhatunk a programlista készítése közben. Ezen tevékenységek végrehajtásához a PAGE direktíva a következő módokon adható meg:

PAGE [*sorszám*]**,***sorszélesség*
PAGE +
PAGE

A "*sorszám*" az egy lapon kiírható sorok maximális számát, a "*sorszélesség*" az egy sorban kiírható karakterek maximális számát határozza meg. A sorok száma a [10,255] intervallumba kell, hogy essen. Alapértelmezés szerint egy lapra 50 sor kerül. A sorszélességnek a [60,132] intervallumba kell esnie. Alapértelmezés szerint egy sorba 80 karakter kerül. A sorszélességet úgy is megadhatjuk, hogy a sorok számát nem írjuk ki, vagyis az aktuális beállított sorszámot kérjük a programlista készítéséhez.

A vessző megadása azonban ebben az esetben is kötelező. Ha a PAGE direktíva másodikként felírt alakját használjuk, akkor a szakasz száma 1-gyel nő és a szakaszon belüli lapszám 1-re állítódik be. A program listázásakor a lapok számozásának alakja:

szakasz száma - szakaszon belüli lapszám

Alapértelmezés szerint a lapok számozása 1-1-gyel kezdődik. A PAGE direktíva argumentum nélküli formájával lapemelést hajthatunk végre programlista készítése közben. A lista-file-ba lapemelés karakter kerül, valamint cím-et és alcím-sort generál.

Példák:

```
PAGE 40,121
```

Hatására a programlista egy lapjára 40 sor, egy sorába 121 karakter kerül.

```
PAGE ,132
```

Hatására a programlista sorszélessége 132 karakter lesz, az egy lapon kiírt sorok száma változatlan marad.

```
PAGE +
```

A direktíva növeli a szakasz számát és a szakaszon belüli lapszámot 1-re állítja.

```
PAGE
```

Lapemelést hajt végre.

5.4.5 A .LIST és a .XLIST direktíva

A .LIST és a .XLIST direktívával azt határozhatjuk meg, hogy mely forrás-sorok jelenjenek meg a programlistában. A .LIST illetve a .XLIST direktíva alakja:

```
.LIST
.XLIST
```

A .XLIST direktíva megszünteti a további forrás-soroknak a listába való helyezését. A .LIST direktíva hatására a direktíva utáni utasítások bekerülnek a listába. Ezeket a direktívákat általában együttesen használjuk abból a célból, hogy egy adott forrás-file valamely része ne jelenjen meg a program listájában.

A .XLIST direktíva minden egyéb, listázással kapcsolatos direktívát felülbírál.

5.4.6 A .LALL, a .XALL és a .SALL direktíva

A .LALL, a .XALL és a .SALL direktívával a forrás-file-ban kifejtett makrók utasításainak listázását vezérelhetjük. A direktívák alakja:

```
.LALL
.XALL
.SALL
```

A makró-assembler mindig listázza a teljes makró-definíciót, de a makró-kifejtés listázását a programozó határozhatja meg. A .LALL direktíva hatására a makró-assembler a makró-kifejtés minden utasítását listázza a makró-megjegyzések (;;szöveg) kivételével. A normál assembly megjegyzések (;szöveg) megjelennek a listában. A .XALL direktíva hatására a makró-kifejtésből csak azok az utasítások kerülnek a listába, amelyek adatot vagy kódot generálnak. Így a makró-assembler a megjegyzéseket és pl. a változó = változó + 1 utasításokat nem listázza. A .SALL direktíva elnyomja a makró-kifejtés összes utasításának listázását.

Ha egyik direktívát sem adjuk meg, akkor .XALL az alapértelmezés.

5.4.7 A .SFCOND, a .LFCOND és a .TFCOND direktíva

A .SFCOND, a .LFCOND és a .TFCOND direktívával feltételes blokkok listázását vezérelhetjük. A direktívák alakja:

```
.SFCOND
.LFCOND
.TFCOND
```

A .SFCOND direktíva hatására a fordító nem listázza a továbbiakban azokat a feltételes blokkokat, amelyekben az IF feltétel nem teljesül. A .LFCOND direktíva visszaállítja ezeknek a blokkoknak a listázását.

A .TFCOND direktívával a feltételes blokkok listázási módjának az alapértelmezését állíthatjuk be. A .TFCOND direktíva hatása attól függ, hogy a fordító hívásakor megadtuk-e a /X opciót (ld. VII. fejezet 2. pont). Ha igen, akkor a .TFCOND direktíva elnyomja az olyan feltételes blokkok listázását, amelyekben a feltétel nem teljesül. Ha a fordító hívásakor nem adtuk meg a /X opciót, akkor a .TFCOND direktíva hatására a "hamis" feltételes blokkok listázása lesz az alapértelmezés.

Példák:

```
.SFCOND
IF 0
        ; Nem listázza az utasításokat
ENDIF
.LFCOND
IF 0
        ; Listázza az utasításokat
ENDIF
```

5.4.8 A .RADIX direktíva

A .RADIX direktíva segítségével beállíthatjuk a forrás-file-ban lévő számok alapértelmezés szerinti számrendszerét. A .RADIX direktíva alakja:

.RADIX *alapszám*

Az "*alapszám*" azt határozza meg, hogy a forrás-file-ban használt számok milyen számrendszerben értelmezendők. A számrendszer alapja 2-16 lehet. Általában a következő értékeket szokás használni:

2 bináris
8 oktális
10 decimális
16 hexadecimális

A .RADIX direktívában megadott "*alapszám*"-ot mindig decimális számként értelmezi a fordító, attól függetlenül, hogy mi az aktuális alapértelmezés szerinti számrendszer.

Példák:

```
.RADIX 8
.RADIX 10
```

A .RADIX direktíva hatása a DD, DQ és DT direktívákra nem terjed ki. A DD, DQ és DT direktívákban megadott számokat a fordító mindig decimális számoknak tekinti. A .RADIX direktíva hatása a B és D opcionális típusjelek megadása mellett sem érvényesül. Ha valamely egész szám után a B vagy D típusjel áll, akkor a számot a B (bináris) vagy D (decimális) számrendszer által megengedett jegyekkel kell felírni, abban az esetben is, ha az aktuális alapértelmezés szerinti számrendszer hexadecimális. Például

```
5ACB
```

numerikus érték hibás, mert 5, A és C nem bináris számjegyek.

5.4.9 A .CREF és a .XCREF direktíva

Az .CREF és a .XCREF direktívával a fordító keresztreferencia-file-jának előállítását befolyásolhatjuk. A direktívák alakja:

.CREF
.XCREF [*azonosító*[,,,]]

A .XCREF direktíva hatására a fordító nem készít keresztreferenciát a címkékről, változókról és egyéb szimbólumokról. A .CREF direktíva visszaállítja a fordító keresztreferencia-generáló tevékenységét.

Ha a .XCREF direktívában azonosítókat is felsorolunk, akkor az itt megadott *azonosítók* **kivételével** minden címke, változó és egyéb szimbólum bekerül a keresztreferencia-fileba.

5.4.10 A NAME direktíva

A NAME direktívával az aktuális modul nevét határozhatjuk meg. Alakja:

NAME *modulnév*

A *modulnév* betűk és számjegyek tetszőleges kombinációja. Bármilyen hosszúságban megadható, de csak az első 6 karakter szignifikáns. A modulnévnek egyedinek kell lennie és nem lehet foglalt név.

A modulnevet a futtatható programot előállító szerkesztő (lásd a 8. fejezetet) használja, amikor hibajelzéseket ír ki.

Ha a NAME direktívát nem használjuk, akkor a fordító a TITLE direktívával megadott szöveg első 6 karakterét tekinti modulnévnek. **Abban az esetben, ha TITLE direktívát sem adtunk meg, az alapértelmezés szerinti modulnév "A".**

5.4.11 A %OUT direktíva

A %OUT direktíva hatására a fordító egy üzenetet jelenít meg a felhasználó terminálján. A direktíva alakja:

%OUT *szöveg*

A direktíva használata főleg hosszú programok fordításakor célszerű, mivel így információt kaphatunk a fordítási folyamat előrehaladásáról.

A %OUT direktíva outputot állít elő mindkét fordítási menetben. Az IF1 és IF2 direktívát annak eldöntésére is használhatjuk, hogy kiírjuk-e a képernyőre az üzenetet.

Példa:

```
IF2
  %OUT   Már a második menetnél tart
ENDIF
```

5.5 Magasszintű nyelvből hívott assembly rutinok

A legtöbb magasszintű nyelv fordítója lehetőséget biztosít a felhasználó által írt assembly nyelvű rutinok beépítésére. Az assembly rutin alkalmazásának szabályai **nem egységesek**, de az adott fordító felhasználói kézikönyvében mindig találunk ismertetést az assembly rutinok használatának feltételeiről. Ilyen feladatra csak gyakorlott assembly-programozók vállalkozzanak!

Mindig pontosan nézzünk utána a belső adatábrázolási formátumoknak és a paraméterátadási mechanizmus működésének. A paraméterátadás általában a vermen keresztül történik (de ez nem törvény), a paramétereknek vagy az értéke, vagy a címe férhető hozzá az assembly rutinból. A regisztereket belépéskor mentsük el, és visszatérés előtt állítsuk vissza (kivéve, ha a paraméterátadás regiszterben történik). Különösen fontos tisztázni, hogy a vermet milyen állapotban kell visszaadni (az ott kapott paramétereket üríteni kell vagy nem, a visszaadandó paramétereket hogy kell elhelyezni... stb)!

Az elkészített assembly rutint teszteljük le még assembly környezetben, csak ezután próbáljuk meg a magasszintű nyelvbe beilleszteni!

6. A MASM fordító használata

6.1 Bevezetés

A **Microsoft Macro Assembler** (MASM) az assembly nyelvű forrásprogramokat áthelyezhető tárgyprogrammá (.OBJ) alakítja, amelyekből a LINK szerkesztőprogrammal készíthetünk végrehajtható (.EXE) file-t, amely az operációs rendszerből közvetlenül elindítható.

6.2 Forrás-file fordítása

A szövegszerkesztővel megírt forrásfile-okat a MASM segítségével fordíthatjuk le. A fordítót a következő paranccsal indíthatjuk el:

MASM [[*kapcsolók*]*forrás*,[*tárgy*],[*lista*],[*crf*]]

kapcsolók	a fordító működését befolyásoló paraméterek, leírásuk a 3. pontban található
forrás	a forrásszöveget tartalmazó file neve
tárgy	a tárgykód-file neve (ez lesz a fordítás eredménye)
lista	a lista-file neve
crf	a keresztreferencia file neve

A parancssorban akár minden paraméter elhagyható, ekkor a MASM kérdéseket tesz fel minden megadható paraméterre vonatkozóan.

Amikor paraméterrel indítjuk el az assemblert, a forrás megadása kötelező és a többi filenév elhagyható (de a vesszők nem!), az elhagyott nevek helyett a rendszer a forrás nevét helyettesíti be, de csak a név helyett mert a kiterjesztésekre konvenciókat alkalmaz (ha nem adunk meg kiterjesztést, mindig helyettesítődik az alapfeltételezés). A konvenciók és alapértelmezések a következők:

forrás	a kiterjesztésre **.ASM**
tárgy	file-név azonos a < forrás > -sal, kiterjesztése **.OBJ**
lista	file-név **NUL** (nem készül lista), kiterjesztése **.LST**
crf	file-név **NUL** (nem készül keresztreferencia) kiterjesztése **.CRF**

Ha a parancssorban bármelyik vessző helyén pontosvesszőt írunk, a MASM a további paraméterek helyén az alapfeltételezést helyettesíti.

Példák:

MASM forr;

Az assembler program a FORR.ASM file-ban van, FORR.OBJ lesz az object file neve, lista és kereszthivatkozási lista nem készül.

MASM forr,,;

Az assembly program a FORR.ASM file-ban van, FORR.OBJ lesz az object és FORR.LST a lista file neve, kereszthivatkozási lista nem készül.

```
MASM forr.src,forr.trg,lista.lis,xr.xrf
```

Az assembler program a FORR.SRC file-ban van, az object a FORR.TRG file-ba, a lista a LISTA.LIS nevűbe és a kereszthivatkozás az XR.XRF file-ba kerül.

Ha nem adtunk meg minden paramétert, de nem pontosvesszővel zártuk le a sort, a hiányzó paraméterekre rákérdez a MASM ugyanúgy, mint amikor egyáltalán nem adunk paramétert a fordító indításánál. A kérdések a következők:

```
Source filename [.ASM]:forr
Object filename [forr.OBJ]:<ENTER>
Source listing  [NUL.LST]:<ENTER>
Cross reference [NUL.CRF]:<ENTER>
```

Az alapértelmezések behelyettesítése most is megtörténik, a fenti példában az assembly program a FORR.ASM file-ban van, az object a FORR.OBJ-be kerül, lista és kereszthivatkozás nem készül.

Bármelyik filenév kiegészíthető meghajtó azonosítóval és elérési úttal is. Kapcsolók bármelyik sor végén megadhatók.

Hasonlóan a parancssorhoz, ha az .ASM file megadása után bárhol pontosvesszőt írunk, a további kérdéseket nem teszi fel a MASM és az alapértelmezéseket behelyettesíti.

6.3 A MASM kapcsolói

6.3.1 /D, az első menet listája

A kapcsoló hatására a fordítás mindkét menetében listát készít a MASM, amiben a hibajelzések is szerepelnek. Erre akkor van szükség, ha a fázishiba (Phase Error) okát nem találjuk. Első menetbeli lista csak akkor készül, ha a lista-file-t megadjuk, ha nem, csak az első menet hibajelzései íródnak ki (azok is amelyeket a második menet felold!).

6.3.2 /ML, kis- és nagybetűk megkülönböztetése a lokális szimbólumokban

Ha ezt a kapcsolót nem adjuk meg, a MASM minden szimbólumot nagybetűsre alakít. Ha ez a tulajdonság nem kívánatos, ezzel a kapcsolóval kikapcsolhatjuk a konverziót. Ebben az esetben a 'Cimke' és a 'cimke' azonosító különböző lesz!

6.3.3 /MX, kis- ésnagybetűk megkülönböztetése a globális szimbólumokban

Ha ezt a kapcsolót nem adjuk meg, a MASM minden globális szimbólumot nagybetűssé alakít. Ha ez a tulajdonság nemkívánatos, ezzel a kapcsolóval kikapcsolhatjuk a konverziót. Erre akkor van szükség, ha olyan tárgymodullal akarjuk összeszerkeszteni programunkat, amely megkülönbözteti a kis- és nagybetűket.

6.3.4 /X, a feltételes fordítás hamis ágának listázása

Ha ezt a kapcsolót nem adjuk meg, a MASM azokat a forrássorokat nem listázza, amelyek egy feltétel hamis ágában vannak. Ez az IF utasítás minden formájára igaz. Amikor a kapcsolót megadjuk, a hamis ágak forrássorai bekerülnek a listába.

Az X kapcsoló hatását befolyásolják a forrásszövegben elhelyezett direktívák is, ezek:

.SFCOND .LFCOND .TFCOND

A .SFCOND és a .LFCOND az X kapcsolótól függetlenül vezérlik a listázást. A .TFCOND minden előfordulása ellenkezőjére változtatja az eddigi hatását az X kapcsolónak (első .TFCOND után a hamis ágakat nem listázza, a második után újra listázza... stb.).

direktíva	/X hatása
.SFCOND	hatástalan, nem listáz
.LFCOND	hatástalan, listáz
.TFCOND	nem listáz
nincs	listáz

Mindez természetesen csak akkor igaz, ha kértünk listát a fordítótól (megadtuk a lista-file nevét).

6.3.5 /R, kódképzés az aritmetikai processzor számára

Ha a direktívát megadjuk, a fordító a 8087/80287 aritmetikai processzor mnemonikjait utasításokká fordítja. Az így fordított program csak olyan gépen fut, amelyben van aritmetikai processzor!

6.3.6 /E, kódképzés a lebegőpontos emulátor számára

Ha a kapcsolót megadjuk, a 8087/80287 aritmetikai processzor mnemonikjai helyett olyan hívásokat generál a fordító, amelyek 8086-ot utasításokkal emulálják az aritmetikai processzor műveleteit. Ezeket a rutinokat egy különálló emulátorkönyvtár tartalmazza, ezt hozzá kell szerkeszteni a programhoz, mert ha nem tesszük, nem definiált szimbólumok keletkeznek a programban. Emulátor könyvtárat a magasszintű programnyelvek programcsomagjai általában tartalmaznak.

6.3.7 /A, szegmensek abc sorrendben

Amikor a kapcsoló nincs megadva, a szegmensek a tárgykódbeli előfordulásuk sorrendjében kerülnek lefordításra, a kapcsoló megadásával a szegmensnevek szerinti abc sorrendben.

6.4 A lista-file

Ha a lista-file nevét megadjuk a MASM indításakor, a forrásprogramról részletes listát készít a MASM. A listában szerepel minden utasítás címe, tárgykódja és maga az utasítás is. A forráslista végén kigyűjtött listát kapunk a használt makrók, struktúrák, rekordok,

szegmensek, csoportok és egyéb szimólumok azonosítóiról és jellemzőiről. Az azonosítók csoportonként névsorba vannak rendezve.

6.4.1 A forrássorok

A forrássorok alakja a listában:

[*sorszám*] *offset kód utasítás*

sorszám a forrássorok 1-től kezdődő folyamatos sorszáma, csak akkor szerepel ha .CRF file-t megadtunk

offset az utasítás szegmensen belüli címe

kód az utasításnak megfelelő object kód vagy adat. Amennyiben ez nem végleges értékű, megadja a kiszámítás módját.

utasítás egyszerű esetben a forrásszöveg egy sora, makró-kifejtés esetén ez eltérhet a forrásszövegtől

A megtalált hibákat a hibás sor után írt hibajelzéssel jelzi a MASM.

A "kód" mezőben használt speciális jelölések a következők:

jel	**jelentése**
R	relokálandó cím, végleges értékét a szerkesztő számítja ki
E	külső (EXTRN) cím, értékét a szerkesztő adja meg
-	szegmens (csoport) cím, értékét a szerkesztő számítja ki
=	EQU vagy = direktíva
nn:	az utasítás szegmensmódosítást ír elő
nn/	az utasításban REP vagy LOCK prefix van
nn[xx]	DUP kifejezés
+	a sor makrókifejtésel keletkezett
C	a sor egy INCLUDE file sora

6.4.2 A makrótáblázat

Abc sorrendben felsorolja a defíniált makrókat és a makródefiníció forrássorainak számát.

6.4.3 A struktúra- és rekordtáblázat

A táblázat tartalmazza a programban előforduló struktúrák, rekordok és rekordmezők azonosítóit abc sorrendben.

A struktúra azonosító sorában a Width oszlop a struktúra mérete byte-okban, a # fields Width oszlopban a mezők száma található. A struktúra mezőazonosítók sorában a shift oszlop a mező távolságát adja byte-okban a struktúra kezdetétől.

Rekord esetén a rekordazonosító sorában a Width oszlopban a rekord mérete szerepel bitekben, a # fields Width oszlop a mezők számát adja.

A mezőazonosítók 'Shift' oszlopa a mező első bitjének távolságát adja a rekord legkisebb helyiértékű bitjétől számítva. A # fields Width oszlop a mező szélessége bitekben. A Mask oszlop a mező maszkjának értéke. Az Initial oszlop a mező inicializálási értéke.

6.4.4 A szegmens- és csoporttáblázat

A táblázat oszlopai:

Name	a szegmens, csoport neve
Size	mérete byte-okban
Align	az igazítás típusa
Combine	a kombináció típusa
Class	a kombinációs osztály neve

Az igazítás típusa BYTE, WORD, PARA és PAGE lehet. Ha nem adtuk meg, PARA lesz. A kombináció típusa PUBLIC, STACK, MEMORY és NONE lehet. Ha nem adtuk meg, NONE lesz. A kombinációs osztály neve az a név, amit a szegmens definíciójában megadtunk.

6.4.5 A szimbólumtáblázat

A programban használt szimbólumok abc sorrendbe rendezett listája, új kezdőbetűnként egy üres sorral elválasztva. A táblázat oszlopai:

Name	név
Type	típus
Value	érték
Attr	tulajdonságok

A típus oszlopban a következők szerepelhetnek:

L NEAR	közeli címke
L FAR	távoli címke
L BYTE	byte adat cimkéje
L WORD	szó adat cimkéje
L DWORD	dupla szó adat cimkéje
L QWORD	4 szó adat cimkéje
L TBYTE	10 byte adat cimkéje
N PROC	közeli eljárás
F PROC	távoli eljárás
Number	szám
Alias	egy szimbólum másik változata
Opcode	utasításkód
Text	karakter (sor) konstans

Az érték oszlopban a szimbólum értéke szerepel.

A tulajdonság oszlopban a definiáló szegmens neve, a szimbólum hatásköre (External, Global) vagy eljárás esetén a mérete byte-okban.

6.4.6 Az első menet listája

Ha a /D kapcsolóval előírtuk, a MASM a fordítás első menetében is listát készít. Az első menet listája tele lesz hibajelzésekkel, mert pl. minden előrehivatkozás cimkéje az első menetben a fordítás pillanatában ismeretlen címke, és ezt hibaként jelzi is a fordító. Ez a hiba persze a második menetben megszűnik, hisz akkor már ismert lesz a címke. Makacs fázishibák esetén sokat segíthet ez a tulajdonsága a fordítónak, mert a két fázisban eltérő lesz a kapott hibajelzés.

Pl: egy mintaprogram:

```
code  segment
      jmp cimke
cimke label fare
      cmp al,1
code ends
```

Ezt lefordítva két hibajelzést kapunk:

```
     jmp cimke
Error...57:Illegal size for item

     cimke label far
Error...6: Phase error between passes
```

Most első menetben is listát kérve csak egy hiba lesz:

```
     jmp cimke
Error...9:Symbol not defined
```

Az első menetben a cimke definiálatlan, mert előrehivatkozás, de a cimke típusa nincs megadva, ezért a near alapértelmezést rendeli hozzá a fordító és két byte helyet foglal számára az ugró utasításban. A második menetben már ismert a cimke és tudjuk, hogy far típusú, de az ilyen cimkére ugrás 4 byte-ot igényel és két byte helye van, ez fázishibát okoz. A hiba egy lehetséges feloldása:

```
jmp far ptr cimke
```

7. Programkönyvtár-kezelés a LIB segédprogrammal

7.1 A programkönyvtár és kezelése

A programkönyvtár háttértáron - általában lemezen - tartott programgyűjtemény. Egy gyűjtemény programjai különféle, az adott gépen rendelkezésre álló magasszintű nyelven és assemblerben készültek, és vagy eredeti forrásszöveg alakban, vagy már lefordított, tárgykódú változatban (utóbbiakat nevezik tárgymoduloknak vagy object moduloknak) szerepelnek a gyűjteményben. Ez a fejezet a tárgymodul könyvtárakkal foglalkozik.

A programkönyvtár fenntartásának olyan operációs rendszerek használatakor van elsősorban értelme, amelyek tartalmaznak rendszerprogramokat

- a könyvtárbeli programoknak a felhasználói programokkal való összeépítésehez, és
- a programkönyvtár karbantartásához.

A PC DOS programszerkesztője, a LINK teljesíti az első követelményt.

Az operációs rendszerhez rendelkezésre áll egy programkönyvtárkezelő, a **Microsoft Library Manager**, a LIB. A LIB tárgymodulokat kezel, és biztosítja a szokásos lehetőségeket:

- a programkönyvtár létrehozása
- a könyvtár programjairól katalógustáblázat létrehozása és karbantartása a könyvtár változásai szerint
- e katalógus kinyomtatása a közösnek deklarált szimbólumokkal együtt
- a könyvtár karbantartása, azaz az egyes programok, modulok *felvétele* a könyvtárba (add), *törlése* (del), *kicserélése* (replace), a felhasználói programokhoz való továbbítása (copy), kérésre a könyvtárból való törléssel együtt (move).

A karbantartó funkciók, akárcsak a PC DOS-hoz mellékelt más rendszerprogramok esetében is, többféle módon elérhetők:

- a LIB hívása után a LIB promptjain
- parancs-soron
- @ file-ba eltett parancsgyűjteményen
- a fenti lehetőségek kombinációján keresztül.

A fejezetben ismertetjük a LIB-hez e három módon való hozzáférés szabályait, és a könyvtár, valamint a tárgymodulok előbb felsorolt kezelési lehetőségeit.

7.2 A LIB funkciói

A LIB funkcióihoz háromféle módon férhetünk hozzá:

- a LIB parancs hívásakor átadott paraméterekkel;
- párbeszédes üzemmódban a LIB kérdéseire adott válaszokkal;
- file-ban előre összeállított parancssorozattal.

A LIB működés közben bármikor megálllítható a <Ctrl + C> billentyűk lenyomásával.

A funkciók a könyvtár módosítására szolgálnak. A könyvtár könyvtármodulokból áll, amelyek kiterjesztése alapértelmezés szerint .OBJ. A könyvtárat csak tárgymodulokkkal lehet bővíteni. A könyvtár változtatására szolgáló funkciók hívása: a funkció jelét, amely egy adott modulra vonatkozik, közvetlenül (szóköz nélkül) a kérdéses modul neve elé írjuk. A "műveleti" jelek és a hozzájuk tartozó funkciók:

Jel	A funkció feladata
+	Egy modul hozzáadása (ADD)
-	Egy modul törlése (DELETE)
-+	Egy modul helyettesítése azonos nevű modullal (REPLACE)
*	Egy modul átmásolása az aktuális tartalomjegyzékbe (EXTRACT)
-*	Egy modul átírása az aktuális tartalomjegyzékbe, és törlése a könyvtárból (EXTRACT and DELETE)
+	Egy könyvtár hozzáadása (a kiterjesztés .LIB) (JOIN)

Az egyes parancsokra rendre bemutatunk egy-egy példát:

A kiindulási állapot:

Az aktuális lemezegység a C. Ezen van a LIB segédprogram és egy MKT.LIB nevű könyvtár, valamint a következő modulok:

X.OBJ, A.OBJ, B.OBJ, C.OBJ, és T.OBJ.

A C lemezegységen van egy MUNKA tartalomjegyzék, benne egy MOD.LIB nevű könyvtár, melynek elemei a következő modulok:

R.OBJ, S.OBJ, T.OBJ, U.OBJ.

A T.OBJ nevű modulok ugyanannak a rutinnak a változatai. A MUNKA-beli T.OBJ az újabb. A parancsok a táblázatban felírt sorrend szerint:

LIB c:\MUNKA\MOD + X

Hatása: a MUNKA-tartalomjegyzékbeli MOD könyvtárunk kiegészül az "X" modullal azaz tartalma R,S,T,U,X lesz.

LIB C:\MUNKA\MOD -U

Hatása : a MOD.LIB tartalma R,S,T,X lesz, az U modul törlődik a könyvtárból.

LIB \MUNKA\MOD - + T

Hatása: A MOD.LIB-ben levő T.OBJ rutint újra cseréli.

LIB \MUNKA\MOD *R

Hatása: a MOD.LIB-ből az R.OBJ modult kiírja az aktuális tartalomjegyzékbe.

LIB \MUNKA\MOD -*S;

Hatása: az S.OBJ modult törli és kiírja az aktuális tartalomjegyzékbe. Így a MOD.LIB tartalma R,T,X lesz.

LIB \MUNKA\MOD +X-U-+T*R-*S+MKT.LIB;

A LIB ilyen esetben először elvégzi a file-kiírásokat (a * művelet), majd az összes törlést, azután veszi sorra a többi műveletet.

7.2.1 A LIB hívása parancs-sorral

A LIB indítása PC DOS parancs-sorral a következő formában történik:

LIB *könyvtár-file neve*[*,kit*][**/PAGESIZE:***n*] *parancs*[*,lista-file*][*,output-file*]

A könyvtár-file az a programgyűjtemény, amelyet a LIB-bel kezelünk. Ha kiterjesztését nem adjuk meg, akkor az alapértelmezés .LIB.

A **/PAGESIZE** opció a könyvtár lapméretét határozza meg, amely alapértelmezés szerint az újabb verziókban 16 byte. (A lapméretre állítás lehetőségére még visszatérünk.)

Az egy vagy több "*parancs*"-csal azokat a műveleteket - LIB funkciókat - adjuk meg, amelyeket a könyvtár file-on végre akarunk hajtatni.

A "*lista-file*" a keresztreferencia-lista file neve. Ilyen file csak akkor készül, ha ezt a paramétert megadjuk.

Az "*output-file*" megadására akkor van szükség, ha más néven akarjuk megőriztetni a könyvtár-file új, a LIB végrehajtása utáni formáját. Ha ez a paraméter elmarad, a módosult könyvtár-file neve ugyanaz lesz, mint az eredetié. Az új könyvtár előállítása előtt a LIB az eredeti könyvtárat átnevezi, a könyvtár régi változata új nevének kiterjesztése .BAK lesz. A file-azonosítók írásában a PC DOS konvenciók érvényesek.

Ügyeljünk a parancsok utáni vesszőkre, és ha csak output file-t adunk meg, akkor „,"-vel jelezzük, hogy nincs lista-file. A parancssor végét bármely elem után ;-vel jelezhetjük, ekkor a további paraméterekre az alapértelmezések érvényesek.

Példa:

LIB rend -+olv;

A parancs hatására a LIB a REND.LIB nevű könyvtárban az "olv" modult az aktuális tartalomjegyzékben található OLV.OBJ modullal helyettesíti. A ; jelzi, hogy a lista- és az output-file-ra az alapértelmezés érvényes, vagyis keresztreferencia-lista nem készül, és a módosítás után az új könyvtár a régi REND.LIB néven kerül tárolásra.

LIB rend -+olv,rend.LST,rend1.LIB

Ezzel a paranccsal a REND.LIB módosítása után létrehozzuk a REND1.LIB könyvtárat, amelyben már az "olv" modul új változata van. Az új könyvtárról REND.LST néven készül keresztreferencialista.

7.2.2 A LIB használata párbeszédes üzemmódban

A LIB működéséhez szükséges információkat a LIB kérdéseire - úgynevezett promptjaira - adott válaszokkal is beírhatjuk.

A LIB párbeszédes módban való használatának lépései:

1. Begépeljük a könyvtárkezelő segédprogram nevét:

LIB

és megnyomjuk az < ENTER > gombot. Ennek hatására a LIB megkérdezi a kezelni kívánt könyvtár nevét (a képernyőn a következő üzenet jelenik meg):

Library name:

2. Az előbbi üzenetre válaszul beírjuk a könyvtár nevét. Ha nem adunk meg kiterjesztést, akkor .LIB az alapfeltételezés. Új könyvtárat nem létező file-név megadásával hozhatunk létre. A LIB visszakérdez, hogy létrehozza-e:

Library file does not exist. Create?

A válasz YES, ha létre akarunk hozni egy új könyvtárat, NO, ha vissza akarunk térni az operációs rendszerhez. Ezután megkérdezi, hogy milyen műveleteket akarunk elvégezni:

Operations?

3. Egy vagy több parancsot írunk be és megnyomjuk az < ENTER >-t. Ha a paranacsok nem férnek el egy sorban, akkor & jel és < ENTER > után a következő sorban folytathatók. Ha csak a könyvtár konzisztenciáját (ennek jelentésére később visszatérünk) akarjuk ellenőrizni, akkor nem adunk meg parancsot.

Az < ENTER > gomb megnyomása után a LIB megkérdezi a keresztreferencia-lista-file nevét:

List file:

4. Ha kérünk keresztreferencia-listát, akkor válaszul a teljes file-nevet írjuk be, mert a kiterjesztésre nincs alapértelmezés. Ha nincs szükség lista-file-ra, elegendő csak az < ENTER > megnyomása. A képernyőn megjelenik az output-file-ra vonatkozó kérdés:

Output library:

5. Ha a módosítás után új néven akarjuk elmenteni a könyvtárat, akkor beírunk egy könyvtárnevet. A helyzet ugyanaz, mint a parancssor esetén: ha adunk output-nevet, a .LIB kiterjesztés itt is alapfeltételezés.
Ha nem adunk nevet, a program akkor is létrehozza az új könyvtárat az eredeti könyvtárnévvel azonos néven, de ezt megelőzően az eredeti könyvtárat átnevezi úgy, hogy nevének kiterjesztése .BAK lesz.

Megjegyzés:

A 2. lépéstől bárhol írhatunk válaszként ;-t. Ekkor a LIB több kérdést nem tesz föl, és az alapértelmezéseket alkalmazza.

A file-azonosítók írására a PC DOS konvenciók érvényesek.

Példa:

```
LIB
Library file:    SZAMOL <ENTER>
Operations:      +OSSZEAD+SZOROZ&<ENTER>
Operations:      +KIVON+OSZT<ENTER>
List file:       <ENTER>
Output Library:  SZAMOL1<ENTER>
```

A korábbi SZAMOL.LIB könyvtárból egy SZAMOL1.LIB nevű könyvtárat készít a beírt 4 modul hozzáadásával, amelyek mindegyike .OBJ kiterjesztésű. E modulok itt is az aktuális tartalomjegyzékből származnak.

```
LIB
Library file:    REND<ENTER>
Operations:      -+OLV;<ENTER>
```

Ennek hatására a LIB a REND.LIB nevű könyvtárban az "olv" modult az aktuális tartalomjegyzékben található OLV.OBJ modullal helyettesíti. A ; jelzi, hogy a lista és az output file-ra az alapfeltételezés érvényes, vagyis keresztreferencia-lista nem készül, és a módosítás után az új könyvtárnak lesz REND.LIB a neve. (A régi REND.BAK néven megőrződik.)

```
LIB
Library file:    REND<Enter>
Operations:      -+OLV<ENTER>
List file:       REND.LST<ENTER>
Output file:     REND1.LIB<ENTER>
```

A LIB promptjaira adott válaszokkal a REND.LIB módosításával létrehozzuk a REND1.LIB könyvtárat, amelyben már az "olv" modul új változata van. Az új könyvtárról REND.LST néven készül keresztreferencia-lista.

7.2.3 LIB parancs-sorozat használata file-ból (válasz-file)

A LIB működése úgy is vezérelhető, hogy az előző pontban leírt, a promptokra adandó válaszokat előre elhelyezzük egy file-ban, az úgynevezett válasz-file-ban. A válasz-file-t szövegszerkesztő programmal (például az operációs rendszerhez adott EDLIN-nel) hozhatjuk létre. E file azonosítója tetszőleges lehet.

A válasz-file felépítése:

könyvtárnév [**/PAGESIZE:***n*]
[*parancslista*]
[*lista-file*]
[*output-file*]

így, rovatokra tagolva. Csak a parancs-listának lehet folytatássora, ilyenkor sorvégi & jelzi, hogy folytatás-sor következik. A ; jel itt is alkalmazható az előzőekben ismertetett módon.

A válasz-file aktivizálása:

LIB *@válasz-file azonosító*

Ennek hatására a LIB elindul. Működés közben a LIB kiírja a képernyőre a válasz-file feldolgozott sorait.

Ha a válasz-file-ban valamely kérdésre nem talál kielégítő választ, kiírja a kérdést a képernyőre, és feleletre vár.

Példa:

Az előző pontban ismertetett egyik párbeszédes példa összeállítása egy válasz-file-ban:

```
szamol.LIB
+osszead+szoroz+kivon+oszt
szamol1.LIB
```

7.3 Új könyvtár létrehozása

A parancs formája:

LIB *könyvtárnév* [**/PAGESIZE:***n*]

Mint láttuk, ha egy létező könyvtár azonosítóját adtuk meg, a LIB úgy tekinti, hogy azt kívánjuk módosítani. Ha valóban új könyvtárat adtunk meg, akkor a LIB a képernyőre kiírja:

Library file does not exist. Create?

A válasz YES, ha létre akarunk hozni új könyvtárat, egyébként NO. A /PAGESIZE opció a könyvtár lapméretét határozza meg, amely alapértelmezés szerint 16 byte.

7.4 A lapméret átállítása (PAGESIZE)

A lapméret kettő hatvány 16 és 32768 között, az alapértelmezés 16. Ez az egész szám a lapméretet byte-okban adja meg.

A modulok a könyvtárban laphatáron kezdődnek a könyvtár elejétől számítva. Másrészt a LIB a program-modulok tárolásakor és viszszakeresésekor olyan indexelést használ, hogy nagyobb lapméreten több modul fér el. E két szempont figyelembevételével kell a lapméretet megválasztanunk úgy, hogy a helykihasználás, - figyelembe véve a könyvtárban elhelyezendő modulok méretét - optimális legyen.

Általában a kis méret az előnyös, kivéve, ha könyvtárunkban sok modul lesz.

A lapméret átállítása történhet parancs-sorral:

LIB *könyvtárnév* **/PAGESIZE:***n*

vagy a "Library name" prompt-ra adott válaszban:

/PAGESIZE:*n*

beírással a könyvtár nevének megadása után.

Az opció nevét röviden

/P:*n*

alakban is megadhatjuk.

7.5 A könyvtár konziszterciájának ellenőrzése

A könyvtár akkor konzisztens, ha a LIB könyvtár-nyilvántartása szerint a könyvtárban lévő programokat el is tudja érni, azaz "megvan az összes modul". Ilyen ellenőrzésre szükség lehet pl. egy lemez vagy lemezrész átmásolása után. Amikor egy könyvtárat készítünk egy vagy több modullal, a LIB automatikusan ellenőrzi a modulok konzisztenciáját. Az ellenőrzési funkció parancssor formájában:

LIB *könyvtárnév*;

interaktív üzemmódban a "Library name" kérdésre adott

könyvtárnév;

válasszal érhető el.

7.6 A keresztreferencia-lista és készítése

A keresztreferencia-lista valójában két listából áll: az egyik a könyvtárban lévő közösen használható (PUBLIC) szimbólumokat és az azokra hivatkozó modulneveket, a másik a könyvtárat alkotó modulok neveit és a bennük levő PUBLIC szimbólumokat tartalmazza.

Keresztreferencia-lista akkor készül, ha a LIB parancssorban való felhasználásakor megadjuk a "lista-file"-t, illetve interaktív üzemmódban a "List file" kérdésre adunk egy file-azonosítót.

8. Futtatható program előállítása - programszerkesztés

8.1 A LINK indítása

A LINK indítására és vezérlésére három lehetőség van:

- PC DOS parancs-sorral
- a LINK kulcsszóval, majd a LINK által kiírt kérdésekre adott válaszokkal
- egy előre összeállított válasz-file-lal.

A LINK működése bármikor leállítható a <Ctrl-Break> lenyomásával. A LINK a megadott tárgy-file-okból, valamint az esetleg szükséges könyvtári file-okból egy relokálható programot készít, amelynek helyigénye legfeljebb 1 Mbyte lehet.

A továbbiakban részletesen leírjuk, hogyan vezérelhető a LINK, és hogyan készíti el a futtatható programot.

8.1.1 Programszerkesztés DOS parancs-sorral

A szerkesztést indító parancs alakja:

LINK [*opciók*]*tárgy-file-ok*,[*futtatható file*],[*map-file*],[*könyvtár-file-ok*]

Az "*opciók*" a LINK működését vezérlő feltételek. Ezek részletes leírása e fejezet 2. pontjában található. A "*tárgy-file*" egy vagy több olyan tárgy-file (.OBJ) neve, amelyből a programot szerkesztjük. Ezeket a tárgy-file-okat valamelyik fordítóprogram - pl. makró-assembler, PASCAL vagy C - készítette. A "*futtatható file*" a LINK által elkészítendő program-file neve. Ha nem adjuk meg, akkor a LINK az első tárgy-file nevét feltételezi futtatható file-névként .EXE kiterjesztéssel. A "*map-file*"-ba kerül az elkészült program térképének listája. Ha nem nevezünk meg ilyen file-t, vagy az opciók között nem adjuk meg a /MAP vagy a /LINENUMBERS opciót, akkor nem készül map-file.

A "*könyvtár-file*" egy vagy több könyvtár-file neve, amely a programhoz szerkesztendő könytári rutinokat tartalmazza. Ha ilyenre nincs szükség, akkor a név helyébe pontosvesszőt kell írni.

A különböző típusú file-neveket vesszővel választjuk el egymástól. A vesszők kiírása akkor is kötelező, ha valamely file nevét nem adjuk meg. Ha egynél több tárgy-file vagy könyvtár-file neve szerepel, akkor a file-neveket legalább egy szóközzel vagy egy "+" jellel kell elválasztani egymástól. Az opciók bárhol lehetnek a parancs-sorban, csak azonos típusú file-nevek között nem.

A parancs-sorban szereplő file-nevekben nem kötelező megadni a kiterjesztéseket, mert a LINK feltételezi rendre az .OBJ (tárgy), .EXE (futtatható), .MAP (térkép) és .LIB (könyvtár) kiterjesztéseket.

Példák:

LINK UJIT.OBJ,UJIT.EXE,UJIT.MAP,UJIT.LIB

A parancs hatására a programszerkesztő az UJIT.OBJ tárgy-file-ból az UJIT.EXE végrehajtható program-file-t állítja elő az UJIT.LIB könyvtár-file-ban lévő megfelelő rutinok hozzászerkesztésével. A program térképe az UJIT.MAP file-ba kerül.

LINK UJIT,UJIT,UJIT,UJIT

A parancs hatása ugyanaz, mint az előzőé, mert a LINK automatikusan hozzáteszi a file-nevekhez az .OBJ, .EXE, .MAP, .LIB kiterjesztést.

LINK UJIT,,UJIT,UJIT

A szerkesztés eredménye a LINK alapfeltételezései miatt ugyanaz, mint az előző két esetben.

LINK INIT + UJIT,UJIT,UJIT,;

Két tárgy-file-ból (INIT.OBJ és UJIT.OBJ) szerkeszti az UJIT.EXE file-t könyvtár használata nélkül, és elkészíti az UJIT.MAP file-t.

LINK X + Y + INIT/PAUSE,,XY,\KAT\ELEM

A LIB program az X.OBJ, Y.OBJ és INIT.OBJ tárgy-file-okból, valamint a "KAT" altartalomjegyzékben lévő ELEM.LIB könyvtár megfelelő rutinjaiból szerkeszti az X.EXE file-t és elkészíti az XY.MAP file-t. A /PAUSE opció hatására mielőtt a LINK elkezdené a futtatható program előállítását, a végrehajtás felfüggesztődik.

8.1.2 Programszerkesztés interaktív üzemmódban

Lehetőség van arra is, hogy a programszerkesztést párbeszédes üzemmódban hajtsuk végre. Ilyenkor az előző pontban ismertetett parancslista elemeit részenként adjuk meg.

A párbeszédes programszerkesztés lépései:

1. Begépeljük a szerkesztő program nevét:

LINK

és megnyomjuk az <Enter> gombot. Ennek hatására a LINK megkérdezi a tárgymodulok nevét. A képernyőn a következő üzenet jelenik meg:

Object Modules [.OBJ]:

2. Az előbbi üzenetre válaszul beírjuk az összeszerkesztendő tárgy-file-ok nevét. Egynél több tárgy-file esetén a file-neveket szóközökkel vagy "+" jellel kell elválasztani egymástól. Ha a file-nevek nem férnek el egy sorban, akkor a sor utolsó karaktereként a "+"-t üssük le, majd nyomjuk meg az <Enter> gombot! Ekkor a LINK megkérdezi a többi tárgy-file nevét. A file-nevekben nem kötelező kiírni a .OBJ kiterjesztést, mert ez az alapfeltételezés.

Ha megadtuk az összes tárgy-file nevét, nyomjuk meg az <Enter>-t! Ennek hatására a LINK a futtatható file nevét kéri:

Run File [file-név.EXE]:

3. Beírjuk a futtatható file nevét és megnyomjuk az <Enter>-t. Ha nem adunk meg file-nevet vagy kiterjesztést, akkor a LINK az alapértelmezést veszi figyelembe (első tárgymodul neve .EXE illetve .EXE). Ilyenkor elég csak az <Enter>-t megnyomni.

A LINK ezután a map-file nevét kérdezi:

List File [NUL.MAP]:

4. Ha készíteni akarunk map-file-t, akkor írjuk be a nevét és nyomjuk meg az <Enter>-t! A .MAP kiterjesztést nem kötelező megadni, mert alapfeltételezés. Ha nem kérünk map-file-t, akkor elegendő csak az <Enter>-t megnyomni. A LINK következő kérdése a könyvtár neve:

Libraries [.LIB]:

5. Valamennyi könyvtár nevét fel kell sorolni, amelyekben olyan rutinok vagy változók vannak, amelyekre a program hivatkozik, de nem dekralálja azokat. A nevek elválasztása szóközökkel vagy "+" jellel történik. A kiterjesztés alapértelmezése .LIB. A könyvtárneveket több sorban is megadhatjuk úgy, mint a tárgy-file-ok neveit. Ha a könyvtarokat valamelyik fordító automatikusan jelzi a LINK-nek, de ezek nem érhetők el az aktuális útvonal(ak)on, elég a LINK-nek megadni azt, hogy a könyvtárak melyik altartalomjegyzékben vannak. Ilyenkor:

[drive:]\út\

alakú választ kell adnunk. (Pl: C:\COMP\MSC\)

Ha könyvtárhasználatra nincs szükség, akkor elég csak az <Enter>-t leütni.

Ezek után a LINK előállítja a végrehajtható file-t.

Megjegyzések:

Ha valamely file másik könyvtárban vagy másik lemezen van, mint az aktuális alapértelmezés, akkor a file nevéhez meg kell adni a megfelelő elérési út (path) vagy lemez nevét. Ha a LINK nem talál meg egy tárgy-file-t, akkor üzen és vár, amíg lemezt cserélünk, ha szükséges.

Minden sor végén lehetőség van opciók megadására.

Bármely üzenetváltás végződhet pontosvesszővel, amelynek hatása: a LINK befejezi a párbeszédet, az elhagyott kérdésekre az alapfeltételezést alkalmazza és elvégzi a programszerkesztést. Ügyeljünk arra, hogy legalább egy tárgy-file neve szerepeljen!

Példa:

Az 1.1 pontban leírt utolsó példa párbeszédes formája a következő:

```
LINK
Object Modules [.OBJ]: x+y+
Object Modules [.OBJ]: init/PAUSE
Run File [x.EXE]:
List File [NUL.MAP]:xy
Libraries [.LIB]:\kat\elem
```

Az első sort a felhasználó írja. A 2. és 3. sor példa a file-neveknek több sorban történő megadására. A 4. sorban láthatjuk a név alapfeltételezését (x), amely a végleges is, mert nem adtunk másikat.

8.1.3 Programszerkesztés előre összeállított válasz-file-lal

A LINK működéséhez szükséges információkat előre elhelyezhetjük egy file-ban, amelyet válasz-file-nak nevezünk. Ekkor a szerkesztés indítása:

LINK @válasz-file-név

A válasz-file neve tetszőleges lehet. A válasz-file tartalma:

tárgy-file-név
[futtatható file-név]
[map-file-név]
[könyvtár-file-nevek]

A file-ok megadási módjára az előzőekben leírt szabályok érvényesek. Minden file-típusnak külön sorban kell szerepelnie. Ha valamelyik file-típus helyett az alapfeltételezést használjuk, akkor a megfelelő sort üresen kell hagyni.

Indítás után a LINK előveszi a válasz-file-ból a megfelelő információkat és automatikusan elvégzi a programszerkesztést. Szerkesztés közben a válasz-file tartalma folyamatosan megjelenik a képernyőn. Ha a LINK hibát talál a nevek olvasásakor, hibajelzést ad és megvárja, amíg beírjuk a helyes választ.

Példa:

Az előző pontban bemutatott párbeszédes programszerkesztési példának megfelelő válasz-file a következő:

```
x+y+
init/PAUSE

xy
\kat\elem
```

Figyeljük meg, hogy a harmadik sor üres! Ha az xy ott volna, akkor azt a LINK a Run File kérdésre adott válasznak tekintené.

8.1.4 Könyvtárak elérési útjainak megadása

A könyvtárak elérési útjainak megadására két módszer lehetséges:

Az első esetben a LINK parancsban lemezegységek és alkönyvtárak felsorolásával megadhatjuk azokat az elérési utakat, amelyeken haladni kell az adott név keresésekor.

A másik lehetőség az, hogy a LINK hívása előtt a PC DOS SET parancsával értéket adunk a LIB környezetleíró változónak. Ez az érték egy vagy több elérési út. Legfeljebb 16 elérési utat adhatunk meg. Az egyes elérési utakat pontosvesszővel választjuk el egymástól. Egy elérési út leírása:

lemezegység:*altartalomjegyzék*

pl. A:\katal1

Ha a lemezegység vagy az altartalomjegyzék hiányzik, akkor az aktuális lemezegység illetve altartalomjegyzék a feltételezés.

Több elérési út megadása esetén a LINK a felírás sorrendjében balról jobbra haladva keresi a megfelelő könyvtárat. Ha például a kt.LIB könyvtár elérési útjai:

A:\kat1;\kat2;B:

akkor először az "A" lemezegység "kat1" altartalomjegyzékében keresi kt.LIB-et. Ha nem találja, ugyanezen a lemezegységen a "kat2" altartalomjegyzék következik, s ha abban sincs, áttér a "B" lemezegységre és annak az aktuális altartalomjegyzékben folytatja a keresést.

Példák:

```
LINK rend,,rend.MAP,A:\kat1\kt.LIB + dcl + B: + C:\kat2\
```

A LINK a "kt.LIB" könyvtárat csak az "A" lemezegység "kat1" altartalomjegyzékben keresi, mert közvetlenül a könyvtárnév előtt adtuk meg az elérési utat. A "dcl" könyvtárat először az "A" lemezegység aktuális altartalomjegyzékében keresi, majd a "B" lemezegység aktuális altartalomjegyzékében, végül a "C" lemezegység "kat2" altartalomjegyzékében.

```
SET LIB = A:\kat1;B:;C:\kat2
LINK rend,,rend.MAP,kt + dcl
```

Ebben a példában a LINK először az aktuális lemezegység aktuális altartalomjegyzékében keresi a "kt" és a "dcl" könyvtárat. Ha nem találja, akkor az "A" egység "kat1" altartalomjegyzékében, majd a "B" egység aktuális altartalomjegyzékében, végül a "C" egység "kat2" altartalomjegyzékében folytatja a keresést.

8.1.5 A map-file szerkezete

A programról készített map-file lényegében a program főbb részeinek a program kezdetéhez relatív elhelyezéséről ad információt, valamint jelzi az esetleges hibákat.

Ha a LINK parancsban megadtuk a /MAP opciót, akkor kilistázza a public szimbólumokat és azok betöltési címeit is.

A map-file-ban lévő szegmens-információ alakja:

```
START     STOP      LENGTH    NAME    CLASS
kkkkkH    vvvvvH    lllllH    x...x   y...y
  .         .         .         .       .
  .         .         .         .       .
  .         .         .         .       .
```

A táblázat első két oszlopa a szegmensek kezdő- és végcímét mutatja 20 bites hexadecimális alakban. Ezek a címek a load modul kezdetéhez relatívek. A load modul feltételezett kezdőcíme 00000H. Ezekből a címekből képezi az operációs rendszer a valódi címeket a program betöltésekor. A harmadik oszlop a szegmens byte-okban mért hosszát adja meg. A negyedik oszlopban van a szegmens neve, az ötödikben a szegmens típusa (pl. CODE-utasítások, DATA-adatok).

A map-file-ban lévő csoport-információ általános alakja:

```
ORIGIN      GROUP
nnnn:n      x...x
  .           .
  .           .
  .           .
```

Az első oszlop a csoport kezdetét, a második a csoport nevét mutatja.

A LINK a program belépési pontjának címét a map-file végén helyezi el.

Ha a LINK parancssorában megadjuk a /MAP opciót, akkor a LINK a public szimbólumokról is elhelyez két táblázatot a map-file-ban. Az egyik táblázatban abc-sorrendben, a másikban betöltési cím szerinti sorrendben szerepelnek a szimbólumok. A címek szegmens:relatív cím alakban kerülnek a map-file-ba. Ezek a load modul kezdetéhez relatív címek. A load modul feltételezett kezdőcíme 00000H.

Ha a LINK parancsban a /HIGH és /DSALLOCATE opció is szerepel és a program mérete az adatokkal együtt nem haladja meg a 64 Kbyte-ot, akkor az előzőekben említett táblázatokban egyes szimbólumok mellett meglepően nagy címet találhatunk. Ez azt jelenti, hogy a szimbólum a program elé kerül.

8.1.6 Ideiglenes lemez-file - VM.TMP

A LINK munkaterülete a memóriában van. Ha ez a terület programszerkesztés közben megtelik, akkor a LINK létrehoz egy ideiglenes lemez-file-t, amelynek neve az aktuális altartalomjegyzékben VM.TMP lesz. Ekkor a LINK a következő üzenetet küldi:

```
VM.TMP has been created.
Do not change diskette in drive, <d:>
```

Az üzenet jelentése:

VM.TMP készült.
Ne cserélje a lemezt a < d: > egységen!

Ez az üzenet a LINK működése végéig érvényes. A szerkesztés végrehajtása után az ideiglenes lemez-file automatikusan törlődik. *Ha a LINK ideiglenes file-t hoz létre, az esetleg létező VM.TMP nevű file-t törli. Ezért ezt a nevet saját célra ne használjuk!*

8.2 A LINK opciói

Az opciók segítségével vezérelhetjük a LINK működését. Minden opció a "/" jellel kezdődik és a LINK parancsban bárhol előfordulhat, csak azonos típusú file-nevek között nem. A gyorsabb írás kedvéért minden opció neve rövidíthető.

Az opciók listája:

/PAUSE	megszakítja a szerkesztést
/MAP	kiírja a public szimbólumok tábláját
/STACK	verem-méret
/CPARMAXALLOC	a helyigény maximuma a tárban
/HIGH	betöltés a lehetséges legnagyobb címtől
/DSALLOCATE	helyfoglalás adatcsoportnak
/OVERLAYINTERRUPT	megszakítás overlay esetén
/LINENUMBERS	forrás-sorok kezdőcímének kiírása
/NOIGNORECASE	kis- és nagybetűk megkülönböztetése
/NOGROUPASSOCIATION	csoport-hozzárendelés törlése
/NODEFAULTLIBRARYSEARCH	könyvtár-alapfeltételezés törlése
/SEGMENTS	szegmensek számának maximuma
/DOSSEG	MS-DOS szegmens-elrendezés használata

8.2.1 Megszakítás a szerkesztés alatt - PAUSE

Alakja:

/PAUSE rövidítve: **/P**

Ezzel az opcióval megszakítható a LINK működése, mielőtt a végrehajtható file-t (.EXE) lemezre írja. Akkor szükséges, ha a kiírás előtt lemezt akarunk cserélni. Ekkor a LINK üzenete:

About to generate .EXE file
Change disks < hit any key >

Az üzenet jelentése:

A .EXE file generálásához cserélje ki a lemez(eke)t és nyomja meg valamelyik billentyűt! Egy billentyű lenyomása után a LINK folytatja munkáját. *Ha VM.TMP nevű file van már a lemezen, akkor ezt a lemezt nem szabad cserélni.*

Példa:

LINK rend/PAUSE,,,\kat\elem

A parancs hatása az, hogy a REND.EXE file lemezre írása előtt a LINK megáll, majd a lemezcsere után folytatja a működést. A REND.EXE file lemezre írása után ismét megáll, lehetővé téve az eredeti lemez visszahelyezését.

8.2.2 Public szimbólumok táblájának előállítása - MAP

Alakja:

/MAP rövidítve: **/M**

Az opció hatására a LINK elkészíti a public szimbólumok táblázatát abc-sorrendben, valamint cím szerint növekvő sorrendben (ld. 8.1.5) és elhelyezi a map-file-ban. Ha a LINK parancsban nem jelöltünk ki map-file-t, akkor a lista-file előtt megadott /MAP opció hatására automatikusan készül ilyen file, amelynek neve az első tárgy-file neve .MAP kiterjesztéssel.

Példa:

LINK rend/MAP,;

ami egyenértékű a

LINK rend.OBJ,rend.EXE,rend.MAP/MAP,;

paranccsal.

8.2.3 A verem-méret meghatározása - STACK

Alakja:

/STACK:*méret* rövidítve: **/ST:***méret*

Hatására a programhoz tartozó verem helyfoglalása "méret" byte lesz. Értéke

1 < = méret < = 65535

lehet. A méret megadható decimális, oktális vagy hexadecimális formában. Az oktális formának 0-val (nullával) kell kezdődnie, a hexadecimális formának 0X-szel.

A LINK általában automatikusan kiszámítja a verem méretét a tárgymodulokban megadott verem-szegmensek méretei alapján. Ezt a számított értéket lehet felülbírálni a LINK parancs /ST opciójával. Ha nincs verem-szegmens a programunkban, akkor a LINK hibajelzést ad. Ezt elkerülendő, minden programban legalább egy vermet definiáljunk!

Példa:

LINK rend/ST:512,;

vagy más alakban

LINK rend.OBJ/STACK:512,rend.EXE;

Mindkét parancs 512 byte helyet biztosít a verem számára a "rend" nevű programban (futtatáskor). Az 512-t megadhattuk volna oktális (01000) vagy hexadecimális (0X200) alakban is.

(Megjegyzés: Ha egy programból .COM kiterjesztésű futtatható file-t akarunk készíteni, nem szabad vermet definiálni benne. Ilyenkor a LINK természetesen hibaüzenetet ad, de utána a .EXE file-t az EXE2BIN programmal átalakíthatjuk .COM formátumra.)

8.2.4 A program helyigényének maximuma - CPARMAXALLOC

A program maximális helyigényét a CPARMAXALLOC opcióval határozhatjuk meg, amelynek alakja:

/CPARMAXALLOC:*szám* rövidítve: **/C:***szám*

Ennek hatására az operációs rendszer "szám" darab paragrafusnyi területet foglal le a memóriában, mielőtt betölti a programot. (Egy paragrafus hossza 16 byte.) Értéke

1 < = szám < = 65535

lehet. A "szám" megadható oktálisan és hexadecimálisan is. Ekkor 0-val, illetve 0X-szel kell kezdődnie.

Ha nem adjuk meg ezt az opciót, a LINK alapértelmezésben 65535-nek veszi a paragrafusok maximális számát. Mivel ez a teljes tár mérete, az operációs rendszer csak annyi helyet biztosít, amekkora a tár legnagyobb összefüggő blokkja.

Ha megadtuk a CPARMAXALLOC opciót, akkor a lefoglalt paragrafusok száma a "szám"-mal egyenlő, és a fennmaradó memóriaterületet más programok használhatják. Abban az esetben, ha kisebb számot adunk meg, mint amennyi helyre a betöltésre kerülő programnak szüksége van, a LINK automatikusan beállítja a helyigénynek megfelelő hosszúságot.

A program által igényelt paragrafusok számának minimuma sohasem kisebb, mint a program kód- és adatszegmensei paragrafusainak együttes száma.

Példa:

LINK rend/C:127,;

azaz

LINK rend.OBJ/CPARMAXALLOC:127,rend.EXE,;

ami 127 paragrafus (2032 byte) helyfoglalást jelent. A 127 helyett írhattuk volna:

oktálisan: 0177,
hexadecimálisan: 0X7F.

8.2.5 A kezdőcím maximumra állítása - HIGH

Az opció hatására a program kezdőcíme a lehető legnagyobb cím lesz a tár szabad területén. Az opció megadási módja:

/HIGH rövidítve: **/H**

Így lehet a programot a szabad hely végében elhelyezni. Ha nem adunk meg ilyen opciót, akkor a betöltés a szabad hely elején kezdődik. Az opció csak a DSALLOCATE paraméterrel együtt használható.

8.2.6 Helyfoglalás az adatcsoportnak - DSALLOCATE

Az opció hatására a DGROUP nevű csoportot tartalmazó szegmensben a csoport a lehető legnagyobb címre kerül. Az opció alakja:

/DSALLOCATE rövidítve: **/D**

A LINK alapértelmezésben az adatcsoport első byte-jának szegmens-relatív címét 0000H-nak adja meg, míg a /D opció hatására a csoport utolsó byte-jához az FFFFH relatív címet rendeli. Ezt az opciót legtöbbször a /HIGH opcióval együtt használjuk, mert így biztosítani tudjuk, hogy a program előtti fel nem használt területtel jobban gazdálkodjunk.

8.2.7 Forrás-sorok kezdőcímének kiírása - LINENUMBERS

Az opció hatására a LINK a program minden forrás-sorából képzett utasítás(ok) kezdőcímét a map file-ba írja. Az opció alakja:

/LINENUMBERS rövidítve: **/LI**

A LINK által így készített információt fel lehet használni magasabb szintű nyelven írt programok forrásszintű nyomkövetéssel való teszteléséhez. Az opció megadásának hatására csak akkor képez a LINK utasítás-kezdőcímeket, ha:

- van map-file-név a LINK parancsban
- a tárgy-file-ban van sorszámozás (csak a magasszintű nyelvek fordítói készítenek ilyet).

Ha nincs a tárgy-file-ban sorszámozás, akkor a LINK figyelmen kívül hagyja a LINENUMBERS opciót. Ha nem adtunk meg map-file-nevet, akkor a LINENUMBERS a MAP opció funkcióját is ellátja.

8.2.8 Kis- és nagybetűk megkülönböztetése - NOIGNORECASE

Az opció hatására a LINK különbözőknek tekinti a kis- és a nagybetűket. Alakja:

/NOIGNORECASE rövidítve: **/NOI**

Ezt az opciót olyankor haszáljuk, ha az összeszerkesztendő tárgy-file-ok között vannak olyanok, amelyekben a kis- és a nagybetűk különbözők. Ezeket a tárgy-file-okat olyan magasszintű nyelv fordítóprogramja (pl. C fordító) készítette, amely különbséget tesz a két betűtípus között.

Alapértelmezésben a LINK azonosnak tekinti a nagy- és kisbetűket, pl. MAP, Map és map azonos szimbólumok.

8.2.9 A könyvtár-alapfeltételezés törlése - NODEFAULTLIBRARYSEARCH

Alakja:

/NODEFAULTLIBRARYSEARCH rövidítve: **/NOD**

A magasszintű nyelvek fordítói a tárgy-file-hoz hozzátehetik azon könyvtárak nevét, amelyek használatát feltételezik. Ez az információ felülbírálható a LINK parancs /NOD opciójával és egy másik könyvtár megadásával.

Tegyük fel például, hogy a "rend" nevű rendezőprogram C nyelven készült és hivatkozunk benne egy C nyelvű programokat tartalmazó könyvtárra, amelyek a rendezéshez szükséges eljárások. Ezután macro-assembly-ben elkészítünk egy ugyanilyen funkciójú könyvtárat, felgyorsítva az eljárásokat. Ekkor a /NOD opcióval és az új könyvtár nevének megadásával a gyorsabb eljárások szerkeszthetők a programhoz.

8.2.10 Csoport eltávolítása a programból - NOGROUPASSOCIATION

Az opció hatására a LINK figyelmen kívül hagyja a csoportot, amikor a cím hozzárendeléseket végzi. Alakja:

/NOGROUPASSOCIATION rövidítve: **/NOG**

Lehetőleg ne használjuk!

8.2.11 Megszakítás overlay esetén - OVERLAYINTERRUPT

Alakja:

/OVERLEYINTERRUPT:*szám* rövidítve: **/O:***szám*

Ezzel az opcióval beállíthatjuk az overlay betöltő rutin megszakítási számát, azaz felülbíráljuk az alapértelmezés szerinti megszakítási számot (03FH). Értéke

0 < = szám < = 255

lehet. A számot decimális, oktális vagy hexadecimális alakban adhatjuk meg. Az oktális számnak 0-val, a hexadecimálisnak 0X-szel kell kezdődnie.

Ügyeljünk arra, hogy ne használjunk olyan megszakítási számokat, amelyeket a hardver használ, illetve nem illeszkednek az operációs rendszer megszakításaihoz!

8.2.12 Szegmensek maximális számának meghatározása - SEGMENTS

Alakja:

/SEGMENTS:*szám* rövidítve: **/SE:***szám*

Az opció hatására a LINK legfeljebb annyi szegmenst dolgoz fel a program szerkesztésekor, amennyi az itt megadott szám. Ha a program ennél több szegmensből áll, akkor a LINK hibajelzést ad és befejezi működését. Értéke

1 < = szám < = 1024

lehet. A maximális szegmensszám decimálisan, oktálisan vagy hexadecimálisan adható meg. Az oktális számnak 0-val, a hexadecimálisnak 0X-szel kell kezdődnie.

Ha a LINK parancsban nincs /SE opció, akkor 128 az alapfeltételezés, azaz a LINK 128 szegmensnek elegendő helyet foglal le a tárban.

8.2.13 Az MS-DOS szegmens-elrendezés alkalmazása - DOSSEG

Alakja:

/DOSSEG rövidítve: **/DO**

Az opció hatására a LINK az MS-DOS operációs rendszer szegmens-elrendezési konvencióinak megfelelően helyezi el a végrehajtható file szegmenseit.

Az MS-DOS szegmens-elrendezési szabályai:

- A "CODE" osztálynevű szegmensek a végrehajtható file elejére kerülnek.
- Közvetlenül a "CODE" szegmens után helyezi el azokat a szegmenseket, amelyek nem tartoznak a "DGROUP" csoportba.
- A "DGROUP"-hoz tartozó valamennyi szegmens a file végére kerül.

8.3 A LINK működése

A LINK a parancsban megadott megfelelő opciók és a forrás-file-okban lévő direktívák alapján elkészíti a végrehajtható file-t a programok kód- és adatszegmenseit összekapcsolva. Az Így kapott "kép" (image) már a program végső formája, ami változatlanul kerül be a tárba, ha a programot futtatni akarjuk.

A LINK számára a szegmensek és a szegmens-csoportok összeszerkesztésének módját a szegmens-attributumokkal adhatjuk meg a forrás-szövegben, továbbá a LINK parancsban lévő információkkal. A SEGMENT direktíva a szegmensek összekapcsolásához, a GROUP direktíva a szegmens-csoportok kialakításához ad információt. Ezekkel a direktívákkal definiálhatjuk a program valamennyi szegmensének sorrendjét és relatív kezdőcímét. Az "elrendezés" típusai:

- csoportok összeállítása
- azonos típusú csoportok összegyűjtése (osztályok)

- címhatárra igazítás (alignment).

A továbbiakban leírjuk, hogy a LINK hogyan kapcsolja össze a szegmenseket, és hogyan oldja fel a hivatkozásokat.

8.3.1 A szegmensek címhatárra való igazítása (ALIGNMENT)

A szegmensben lévő információ típusának megfelelően megadhatjuk a kezdőcímre vonatkozó kikötésünket.

típus	megnevezés	feltétel a címre
BYTE	tetszés szerinti byte címe	egész szám
WORD	egy szó címe	2-vel osztható szám
PARA	egy paragrafus címe	16-tal osztható szám
PAGE	egy lap címe	1024-gyel osztható szám

Mielőtt a LINK egy szegmenst a végrehajtható file-ba másol, megvizsgálja a soronkövetkező byte címét, hogy kielégíti-e a kezdőcím típusának megfelelő feltételt. Ha igen, akkor folyamatosan kimásolja a szegmenst. Ha nem, akkor a feltételnek megfelelő következő határig üres helyet hagy és azután kezdi másolni a szegmenst.

8.3.2 A szegmensek kezdőcímének meghatározása

Amikor a LINK összeállítja a végrehajtható file-t, a soronkövetkező szegmens lemezre írása előtt kiszámítja a szegmens program-relatív kezdőcímét. Ezt a címet a már kiírt szegmensek együttes hossza és a szegmens kezdőcímének típusa határozza meg. A cím két részből áll:

- paragrafuscím (frame number): az első olyan paragrafus program-relatív címe (16-tal osztható), amely a szegmens elejét tartalmazza
- eltolás: a szegmens első byte-jának távolsága a paragrafus kezdetétől.

Az eltolás értéke csak akkor lehet nullától különböző, ha a szegmens kezdőcíme BYTE vagy WORD típusú. A szerkesztésről kért listában a szegmens kezdőcímének első 5 hexadecimális jegye a frame number.

8.3.3 A szegmensek átrendezése

A LINK olyan sorrendben helyezi el a szegmenseket a végrehajtható file-ban, ahogyan felsoroltuk a tárgy-file-okat a LINK parancsban.

Ha azonos osztálynevű szegmenseket talál, akkor azokat egy folyamatos területen helyezi el, így eltérhet a tárgy-file-ok megadási sorrendjétől.

Abban az esetben, ha a szegmens osztályát nem definiáltuk, a "nulla" osztálynév rendelődik a szegmenshez. Ha a LINK több ilyet talál, ezeket is egy folyamatos területre teszi. Az azonos osztályba tartozó szegmensek száma és együttes helyfoglalásuk nem korlátozott. Az utóbbi meghaladhatja a 64 Kbyte-ot.

8.3.4 Kombinált szegmensek

A szegmens attributumai között a forrás-programban a következő kombinációs típusok adhatók meg:

PUBLIC, STACK, MEMORY, COMMON, PRIVATE.

A LINK az azonos nevű és kombinációs típusú szegmenseket egyetlen nagy szegmensbe gyűjti, amelynek mérete legfeljebb 64 Kbyte lehet. Túllépéskor a LINK hibajelzést ad.

A PUBLIC kombinációs típusú, azonos nevű szegmensek előfordulási sorrendben való egymás utáni elhelyezésével egyetlen nagy szegmens készül. Az így kapott szegmens valamennyi elemének címe ugyanahhoz a "frame address"-hez relatív. A LINK megőrzi az egyes szegmensek igazítási attributumát, és a szegmensek bemásolásánál figyelembe veszi.

A STACK vagy MEMORY típusú szegmensek kombinálása a PUBLIC típusúaknak megfelelően történik, de a STACK esetében a mutató kezdőértéke is a végrehajtható file-ba kerül.

A COMMON típusú szegmensek kombinációjának feltétele: azonos szegmensnév és azonos osztálynév. A kombinált szegmens hossza megegyezik a leghosszabb összetevő szegmens hosszával. Valamennyi összetevő szegmens kezdőcíme egyenlő a kombinált szegmens kezdőcímével, így átlapolva biztosítható a közös terület létrehozása.

Ha a leírt négy kombinációs típus egyike sem szerepel, akkor a szegmenst PRIVATE típusúnak nevezzük. Az ilyen szegmensekből nem készül kombinációs szegmens.

8.3.5 Szegmens-csoportok

Különböző osztályokhoz tartozó és különböző kombinációs típusú szegmensekből szegmens-csoport készíthető, amelynek mérete nem haladhatja meg a 64 Kbyte-ot. Ennek előnye, hogy a csoport szegmens-címe valamennyi benne lévő szegmens közös bázisa lesz, amelyhez relatív a csoportban lévő valamennyi cím. A csoport szegmenseinek elhelyezése nem feltétlenül folytonos, lehetnek közöttük más szegmensek is, de az összméret értelemszerűen ezekkel együtt sem haladhatja meg a 64 Kbyte-ot.

8.3.6 Címfeloldás

Ha a LINK a program valamennyi szegmensének kezdőcímét ismeri, akkor feloldja a még definiálatlan címke- és változó-hivatkozásokat. A már korábban elkészített relatív cím és a kezdőcím adja a valódi, program-relatív címet. A címfeloldás négy különböző hivatkozásra vonatkozik:

- rövid
- közeli önrelatív
- távoli szegmens-relatív
- hosszú.

A kiszámított érték mérete a hivatkozás típusától függ. Ha a megengedett mérettől eltérés van, hibajelzést kapunk. Ez például akkor történhet meg, ha egy 16-bites relatív címet akarunk használni egy olyan szegmensben lévő objektum eléréséhez, amely más "frame address"-hez relatív. Akkor is hibajelzést kapunk, ha valamely szegmenscsoporthoz tartozó szegmensek együttes mérete meghaladja a 64 Kbyte-ot.

A rövid hivatkozás - például a JMP utasításban - olyan címkézett utasításra vonatkozik, amely ugyanabban a szegmensben vagy szegmens-csoportban van. Egy ilyen hivatkozás legfeljebb 128 byte távolságra történhet. A relatív cím 8-bites lesz. Ha az előbbi feltételek nem teljesülnek, akkor hibajelzést kapunk.

A közeli önrelatív hivatkozás olyan adatra vonatkozik, amely ugyanabban a szegmensben vagy szegmens-csoportban van. Ez a relatív cím 16-bites. Egyéb hivatkozáskor hibajelzést kapunk.

A távoli szegmens-relatív hivatkozás olyan adatra vonatkozik, amely egy meghatározott szegmensben vagy szegmens-csoportban van, vagy egy adott szegmensregiszterhez relatív. Ez a relatív cím 16-bites. Egyéb hivatkozáskor hibajelzést kapunk.

Hosszú hivatkozás a JMP vagy CALL utasításban fordulhat elő, amellyel egy másik szegmensben vagy szegmens-csoportban lévő utasításra hivatkozunk. A LINK ekkor kiszámítja a 16-bites "frame address"-t és a 16-bites relatív címet. Ha a cím nem számítható, hibajelzést kapunk.

8.3.7 A betöltés sorrendjének vezérlése

A szegmensek alapfeltételezés szerinti betöltési sorrendjétől úgy lehet eltérni, hogy írunk egy álprogram-file-t (dummy), benne a megfelelő ál-szegmensekkel, amelyek osztálynév szerinti sorrendje megegyezik a betöltés kívánt sorrendjével. Az álprogram lefordítása után a kapott tárgy-file-t kell megadni a LINK parancsban első tárgy-file-ként. Ekkor a LINK automatikusan a leírt sorrendben tölti be a megfelelő szegmenseket.

Ha pl. egy programban a szegmensek nevei a következők:

kód, adat, verem, konstans, tár

akkor egy adott sorrendet biztosító álprogram-file felépítése:

```
kod         SEGMENT     'CODE'
kod         ENDS
konstans    SEGMENT     'CONST'
konstans    ENDS
adat        SEGMENT     'DATA'
adat        ENDS
verem       SEGMENT     'STACK'
verem       ENDS
tar         SEGMENT     'MEMORY'
tar         ENDS
```

Az álprogramban minden olyan osztálynevű szegmensnek szerepelnie kell, amilyen a szerkesztésre váró programban előfordul. Ha a felsorolás hiányos, a szegmensek sorrendje eltérhet a kívánt sorrendből. Egy program szerkesztésekor a LINK parancsban megadott első tárgy-file az a file, amelyet az álprogramból készítettünk.

Megjegyzések:

- A magasszintű nyelveken írt programokban a szegmensek sorrendjét a megfelelő rendszer határozza meg, ezért az álprogram használatának nincs értelme.
- Ha a MEMORY szegmenseket a program végére akarjuk szerkesztetni, akkor az álprogram végére egy üres MEMORY szegmenst írjunk:

```
tar SEGMENT MEMORY 'MEMORY'
tar ENDS
```

9. Programok karbantartása a MAKE program segítségével

9.1 Bevezetés

Programok fejlesztése során mindenkinek szembe kell néznie a programok "karbantartásának" problémájával. Nagyobb programok készítésekor elkerülhetetlen, hogy több modulból építkezzünk, viszont ez igen sok kellemetlenséget is okozhat.

Vegyük példaként az alábbi esetet: Programunkat a következő modulokból építettük fel: PROG.ASM, P1.ASM, P2.ASM (vagy más nyelvek esetén .C, .PAS, stb). A futtatható program neve legyen PROG.EXE.

Ha a forrásfile-ok valamelyikén változtatunk, akkor azt le is kell fordítanunk és linkelnünk, hogy az új futtatható file létrejöjjön. Gyakori eset, hogy több forrásfile-t is módosítunk, de elfelejtjük mindet lefordítani. A futó programba így a régi object file szerkesztődik be, és a programunk továbbra sem működik, pedig mi "kijavítottuk".

Ezt a hibát kiküszöböli és egyben sok fáradtságtól is megkímél bennünket a Microsoft MAKE segédprogramja. Segítségével automatizálható a program "karbantartása", azaz a fordítás, könyvtárazás, szerkesztés. A továbbiakban a MAKE program 4.02-es verziójának használatát és tulajdonságait részletezzük.

9.2 A MAKE használata

Először létre kell hoznunk a MAKE számára egy ún. leíró file-t. Ezt tekinthetjük egy speciális BATCH file-nak is, melyet nem a DOS, hanem a MAKE program fog feldolgozni. A leíró file általában feltételeket és parancsokat tartalmaz az alábbi formában:

output-file: *input-file1* [*input-file2*] ...
parancs1
[*parancs2*]
...

E sorok jelentése a következő: Ha az input file-ok valamelyikének létrehozási időpontja későbbi, mint az output file-é (azaz az input file-ok valamelyike megváltozott az output file létrehozása óta), végre kell hajtani a parancso(ka)t.

Amennyiben a leíró file-unk az alábbi sorokat tartalmazza,

```
PROG.EXE: PROG.OBJ P1.OBJ P2.OBJ PROG.LIB
LINK  PROG + P1 + P2,,,PROG ;
```

a program újraszerkesztésére csak akkor kerül sor, ha a legutóbbi szerkesztés óta megváltozott a PROG.OBJ, P1.OBJ, P2.OBJ, PROG.LIB file-ok valamelyike.

9.3 MAKE leíró file létrehozása

A file-t bármely szövegszerkesztővel létrehozhatjuk. A file-név kiterjesztése bármi lehet, általában elhagyjuk.

A leíró file-ban minden feltétel és parancs külön sorba kell kerüljön. Ha a feltétel túl hosszú, akkor azt egy backslash (\) után új sorban folytathatjuk.

A megadott input file-oknak létezniük kell, ellenkező esetben a MAKE nem tudja értelmezni a feltételt és végrehajtani a parancsokat. Az output file-nak nem kell léteznie, hiszen azt például a következő parancs is létrehozhatja.

A megadott parancs tetszőleges, a DOS számára felismerhető parancs lehet.

Az egyes input file-okat (legalább egy) szóköz karakterrel kell elválasztanunk. Az egyes parancsokat egy TAB vagy szóköz előzze meg, és mindegyiket új sorba kell írnunk.

A leíró file-ban több "leírás" (feltétel-parancs pár) is lehet. Az egyes leírásokat legalább egy üres sor válassza el!

A leíró file a fentieken kívül tartalmazhat még megjegyzéseket is. Ezeket mindig a # karakternek kell megelőznie. A megjegyzés kerülhet az input file-ok vagy a parancsok után, vagy egy új sor legelejére (szóköz nem előzheti meg!).

Ez utóbbi esetben a sor csak a megjegyzést tartalmazhatja.

9.4 A MAKE futtatása

A program indítása a következő:

MAKE [*opciók*] [*makródefiníciók*] *leírófile*

Például ha létrehoztuk PROG.EXE programunk karbantartására a PROG.MAK nevű leírófile-t, azt a következőképpen indíthatjuk el:

MAKE PROG.MAK

Ennek hatására a MAKE a PROG.MAK-ban lévő leírásokat sorra veszi, s a leírt feltételek teljesülésétől függően az egyes parancsokat végrehajtja. Végrehajtás során a MAKE felismeri, a DOS környezet-változóit (pl. PATH), s ezt, pl. a file-nevek megadásakor figyelembe vehetjük (azaz a teljes útvonalnevet csak akkor kell megadnunk, ha a megfelelő elérési út nincs a PATH-ban definiálva).

A MAKE abbahagyja a leírófile feldolgozását, ha nem találja valamelyik input file-t, vagy ha egy hívott parancs hibakóddal tér vissza (pl. nem sikerült a fordítás).

9.5 A MAKE opciói

A MAKE futását az alábbi opciókkal módosíthatjuk:

/D	(display date)	Kiírja a megvizsgált file-ok utolsó módosításának dátumát
/I	(ignore)	Figyelmen kívül hagyja az egyes parancsok visszaadott hibakódját, és folytatja a végrehajtást.
/N	(not execute)	Nem hajtja végre az egyes parancsokat, csupán kiírja azokat a képernyőre.
/S	(silently)	A szükséges parancsokat végrehajtja, de azok nem jelennek meg a képernyőn.

9.6 Makro definíciók

A makrok lehetővé teszik, hogy egy konkrét nevet egy szimbolikus névvel (formális paraméterrel) helyettesíthesünk. Ezzel egy adott leíró-file több célra is használható (ugyanazokat a műveleteket más file-okon is végrehajthatjuk) a leírófile újraírása nélkül.

A makródefiníció formája:

szimbolikus-név = konkrét-név

Hivatkozás a makróra:

$(*szimbolikus-név*)

Az ilyen hívások helyére a MAKE a konkrét nevet helyettesíti be. Makrokat a következő helyeken definiálhatunk:

1. A parancssorban, a MAKE hívásakor;
2. A leíró file-ban;
3. A DOS-ban beállított környezetben (SET).

Amennyiben egy definíció több helyen is előfordul a fenti sorrend szerint előbb álló lesz az érvényes. A MAKE három speciális makrót is felismer:

$* az output file kiterjesztés nélküli nevét jelenti;
$@ az output file kiterjesztés nélküli nevét jelenti;
$** az input file-ok teljes listáját jelenti.

9.7 Példa a MAKE használatára

A PROG.EXE nevű programunk az alábbi forrásokra épül:

PROG.ASM, P1.ASM, P2.ASM

A leíró-file az alábbi sorokat tartalmazza:

```
PROG.OBJ :  PROG.ASM  # PROG.OBJ  f gg PROG.ASM-t¢l
        MASM  PROG;

P1.OBJ   :  P1.ASM
        MASM  P1;
P2.OBJ   :  P2.ASM
        MASM  P2;
        LIB   PROG.LIB -+P2;
PROG.EXE :  PROG.OBJ  P1.OBJ  PROG.LIB
        LINK  PROG+P1,,,PROG;
```

A fenti sorok hatására a MAKE elkészíti a futtatható (PROG.EXE) file-t, ha a források bármelyikét módosítjuk, vagy ha a PROG.LIB könyvtár más file-ok miatt megváltozott.

10. Szimbolikus nyomkövető és hibakereső program (SYMDEB)

10.1 Bevezetés

A Microsoft SYMbolic DEBug utility (szimbolikus hibakereső) program végrehajtható (.EXE vagy .COM) file-ok (és bármilyen más bináris file) módosítására, tesztelésére alkalmas. Visszafordítja a programkódot, módosíthatók a regiszterek, töréspontokat tudunk elhelyzeni a programban és lépésenként futtathatjuk a programot. A SYMDEB-bel tesztelhetünk minden olyan programot, amelyek a Microsoft nyelvek konvenciói szerinti lebegőpontos emulációt használják.

A SYMDEB minden olyan funkciót, parancsot ismer, amelyet a DEBUG hibakeresőt használó programozók megszoktak. Legjelentősebb újítása az, hogy a forrásprogram objektumaira szimbólumaikkal hivatkozhatunk. Adatokat ellenőrizhetünk és módosíthatunk, eljárásokat hívhatunk meg úgy hogy nem kell tudnunk hogy hol (milyen fizikai címen) vannak a memóriában. Ehhez a szimbólumtáblát tartalmazó (.SYM) file-t kell előállítanunk a szerkesztőprogrammal előállított .MAP file-ból. Erre szolgál a SYMDEB-hez tartozó MAPSYM nevű segédprogram.

A SYMDEB-bel tartós (állandó) töréspontokat helyezhetünk el a programban. Ezeken áthaladva a programfutás mindig megszakad és ellenőrizhetünk, módosíthatunk adatokat és a programot is, lépésenkénti végrehajtással futtathatunk tovább... stb. A töréspontok kikapcsolhatók, törölhetők és újra visszakapcsolhatók.

A SYMDEB kifejezéskiértékelője használni tudja a szimbólumokat és számokat is. Ismeri az aritmetikai és logikai alapműveleteket, hivatkozhatunk az objektumok szegmens- és offset-címére, portról beolvasott adatokra... stb.

A Microsoft Pascal, C és FORTRAN fordító forrás-szintű hibakeresésére is alkalmas a SYMDEB. Kijelzi a lefordított program gépi kódját és a forrássorokat is. A kétféle kijelzési módot együtt és külön-külön is használhatjuk.

A Microsoft cég legújabb fordítóihoz új hibakereső és nyomkövető programot, az ún. CODEVIEW-t készített. Ez lényegesen többet tud a SYMDEB-nél (pl. teljes képernyő használata). Jelenleg a Microsoft 'C' 4.0-ás, FORTRAN 4.0-ás és Quick BASIC 3.0-ás verziójú fordítókhoz adják.

10.2 A SYMDEB indítása

A SYMDEB-et a következő alakú DOS parancs-sorral indíthatjuk el:

SYMDEB [**/IBM**] [*symfiles*] [[*filespec*] [*args*]]

/IBM	az IBM kompatibilitást jelzi
symfiles	a .SYM szimbólumtáblát tartalmazó file-ok neveit adhatjuk meg
filespec	a végrehajtható program- vagy bináris file neve
args	a végrehajtható program argumentumai, amit a SYMDEB úgy helyez el, mintha a DOS-ból indítottuk volna a programot.

Ha paraméterek nélkül indítjuk a SYMDEB-et, betöltődés után kiírja a bejelentkező üzenetet és a processzor típusát (a 8088-at és a 8086-ot nem tudja megkülönböztetni, IBM XT-n is 8086-ot ír ki).

10.2.1 Indítás program file-lal

Amikor a DOS parancssorban megadjuk egy .EXE vagy egy .COM file nevét is, pl:

```
SYMDEB FILE.EXE
```

a SYMDEB létrehozza a program szegmens prefixet (PSPA) és betölti a programot a rendelkezésre álló legkisebb című szegmensbe. A file mérete (byte-okban) a BX:CX regiszterpárba kerül és a többi regiszter értéke beállítódik a programfile-nak megfelelően.

10.2.2 Indítás szimbólumokkal

A DOS parancssorban megadhatjuk a szimbólum-file(-ok) nevét is. Ekkor a szimbólumtábla is betöltődik. A szimbólum-file-t a MAPSYM segédprogrammal kell elkészíteni a szerkesztés után, használatát a fejezet 2.5 pontja ismerteti.

Példa:

```
SYMDEB FILE.SYM FILE.EXE
```

10.2.3 Paraméterátadás a betöltendő programnak

A DOS parancssorban megadhatunk paramétereket a futtatandó program számára, pl:

```
SYMDEB FILE.SYM FILE.EXE X.DAT/K
```

Ekkor az .EXE (vagy .COM) file neve mögötti karaktereket a SYMDEB elhelyezi a program PSPA-ban úgy, mintha a DOS indította volna a programfile-t.

A program fejlécről további információkat talál a fejezet 5.26. pontjában a NAME parancsnál.

10.2.4 Indítás file nélkül

Amikor a SYMDEB-et paraméterek nélkül indítjuk, létrehozza a PSPA-t, de nem tölt be programot, ezt a LOAD és NAME parancsok segítségével kell megtennünk, ha szükséges. Paraméter nélküli indítás esetén a szegmensregiszterek a szabad memória legkisebb címére (a PSPA-ra) állítódnak be, az IP értéke 0100H lesz és a többi regiszter nullára áll.

10.2.5 Szimbólum file készítése (MAPSYM)

A szimbólikus hibakeresést biztosító funkciók használatához el kell készítenünk a .SYM file-t. Ennek előfeltétele, hogy a szerkesztőprogramot a /MAP kapcsolóval indítsuk el, vagyis létrehozzuk a .MAP file-t. Ha a forrásszintű nyomkövetést is használni szeretnénk a /LINENUMBERS kapcsolót is meg kell adni. Az így létrehozott .MAP file-ból készít a MAPSYM segédprogram .SYM file-t a SYMDEB számára.

A parancs formája:

MAPSYM [-l|/l] *filespec*

filespec a .MAP file neve. Ha nem adunk meg kiterjesztést a.MAP az alapértelmezés.
-l|/l a kétféle megadás egyenértékű és segítségével előírhatjuk, hogy a MAPSYM a konverzió folyamatáról információkat írjon ki a képernyőre

10.3 A kontroll karakterek funkciója

A DOS által biztosított szerkesztési lehetőségek a SYMDEB parancssorában is rendelkezésre állnak.

10.3.1 Parancs megszakítása

Bármelyik SYMDEB parancs végrehajtását megszakíthatjuk a <Ctrl-Break> illetve <CTRL-C> gombok megnyomásával, ekkor visszakapjuk a SYMDEB promptját. A GO paranccsal elindított program csak akkor szakítható meg a <Ctrl-Break>-kel, ha az futás közben ír a képernyőre, vagy adatot kér a billentyűzetről.

Ha a Redirection paranccsal (35. pont) más eszközről vezéreljük a SYMDEB-et, a <CTRL-C> funkció nem működik.

10.3.2 Parancs felfüggesztése

A <CTRL-S> gomb megnyomásával megállíthatunk és továbbengedhetünk minden olyan SYMDEB parancsot, amelyik a képernyőre ír. Ha a Redirection paranccsal (5.31. pont) más eszközről vezéreljük a SYMDEB-et, a <Ctrl-S> hatástalan.

10.3.3 Az NMI használata

Az NMI segítségével bármikor megszakíthatjuk a program futását (végtelen hurok!), de ehhez HW segédeszközre van szükség (pl: IBM Professional Debug Facility). IBM AT-n a System Request segítségével szimulálható az NMI, de ez nem hatásos, ha a program letiltja a megszakításokat.

10.4 A parancsok használata

10.4.1 A parancsok általános felépítése

parancsnév [*paraméter*,,,]

A *parancsnev*ek egy vagy két karakteresek, a *paramétere*k a parancs jellegétől függően címkifejezések, karakterek stb. lehetnek. A kis- és nagybetűk közt a SYMDEB nem tesz különbséget. Ha több paramétert adunk meg, elválasztásuk vesszővel vagy tetszőleges számú szóközzel történhet.

10.4.2 Szimbólumok

A szimbólumok karakterekkel vagy aláhúzással, dollárral, Ő jellel vagy kérdőjellel kezdődhetnek és a szimbólumon belül számjegyeket is tartalmazhatnak. Szimbólum lehet bármelyik regiszter vagy flag neve és a .SYM file-ban szereplő globális szimbólum neve. A szimbólumokban a SYMDEB a kis- és nagybetűket nem különbözteti meg. Ha két azonos nevű szimbólum van a .SYM file-ban, csak az először előfordulót tudjuk elérni. Ha valamelyik szimbólum neve azonos egy regiszter vagy flag nevével, akkor azt a szimbólumot nem tudjuk elérni.

10.4.3 Számok

számjegyek[*r*]

A SYMDEB parancsokban a számokat a fenti alakban adhatjuk meg és egész típusú értéket kell adniuk. A *számjegyek*nek meg kell felelniük az [*r*]-rel megadott számrendszernek. Ha [*r*]-t elhagyjuk, hexadecimális az alapértelmezés.

[r] jelentése:

Y	- bináris
O,Q	- oktális
T	- decimális
H	- hexadecimális

A megadott szám értéke az [0,65535] tartományba eshet. Ha nagyobb, a SYMDEB annyi jegyet hagy el a szám elejéről, hogy teljesítse ezt a feltételt. A vezető nullák száma nem korlátozott.

10.4.4 Címek

[*szegmens*:]*offset*

A címek a fenti alakban adhatók meg. Bármelyik cím lehet kifejezés is, amely szimbólumokat is tartalmazhat. *Ha a szegmenscímet elhagyjuk a SYMDEB az A,G,L,PT,T,U és W parancsokban a CS, minden más parancsban a DS regiszter értékét használja szegmenscímnek.*

10.4.5 Címtartomány

startcím [*végcím* | **L** *hossz*]

Címtartomány megadásakor a fenti forma használható. Megadható a tartomány vége vagy a hossza, de a második rész el is maradhat. A *startcím* és *végcím* a 10.4.4 pontnak megfelelően adható meg. Ha a második részt elhagyjuk, a SYMDEB L80H-t helyettesít be.

10.4.6 Sorszámok

.+ | **-** [*szám*]
.[*filenév*] *szám*
.szimólum [**+** | **-** *szám*]

A sorszámok a fenti formák egyikében adhatók meg és a forrásszöveg sorszámát jelentik.

Az első forma relatív sorszámot ad meg az aktuális sorhoz viszonyítva. Ha a *szám* elmarad, a + jel a file utolsó sorát, a - jel az első sorát jelenti. A második forma abszolút sorszámot jelent. Ha a *filenev*et megadjuk, abban a forrásfile-ban, ha nem akkor a CS:IP-nek megfelelő file-ban. A harmadik forma a forrásszöveg *szimbólum*mal megadott objektumát, illetve az ehhez viszonyított relatív sorszámot jelenti. Ha *filenév* vagy *szám* nem létezik, hibajelzést kapunk.

10.4.7 Karaktersorozatok

'[*karakter(ek)*]'
"[*karakter(ek)*]"

Karaktersorozatok a fenti formákban adhatók meg. Ha a sorozat a határoló jelet is tartalmazza, azt kettőzve kell megadni.

10.4.8 Kifejezések

Minden kifejezés értéke egy 8, 16 vagy 32 bites érték. Kifejezés bármelyik parancs argumentumaként használható, ahol 8, 16 vagy 32 bites értéket kell megadni. Kifejezésekben számok, szimbólumok, címek és operátorok szerepelhetnek.

A használható operátorokat az alábbi táblázatok tartalmazzák, a felsorolás sorrendje egyúttal precedenciát is jelent. Az előbb felsorolt operátor precedenciája nagyobb.

Egyoperandusú operátorok

+	pozitív előjel
–	negatív előjel
NOT	egyes komplemens
SEG	az operandus szegmenscíme
OFF	az operandus eltolási címe
BY	a kifejezés kisebb helyiértékű byte-ja
WO	a kifejezés kisebb helyiértékű szava
DW	a kifejezés duplaszó alakja
POI	a kifejezés értékének megfelelő 4 byte-os mutató
PORT	a kifejezésnek megfelelő című portról beolvasott byte
WPORT	a kifejezésnek megfelelő című portról beolvasott szó

Kétoperandusú operátorok

*	szorzás
/	egész típusú osztás (DIV)
MOD	maradék
:	szegmens módosítás
+	összeadás
-	kivonás
AND	bitenkénti "ÉS" művelet
XOR	bitenkénti "KIZÁRÓ VAGY" művelet
OR	bitenkénti VAGY művelet

Azonos precedencia esetén a kiértékelés balról-jobbra történik. A precedencia zárójelekkel felülbírálható.

10.5 A SYMDEB parancsai

10.5.1 Az Assemble (fordítás) parancs

A [*cím*]

A megadott *cím*től kezdődően **befordítja a 8086/8087 processzorok mnemonikjaival megadott utasításokat**. Ha nem adunk meg *cím*et, a fordítás a CS:IP értéktől kezdődik.

Minden assembly utasítást az ENTER-rel kell lezárni. Kilépni a parancsból úgy lehet, hogy assembly utasítás helyett üres ENTER-t adunk be.

Az utasítások beírásakor a következő szabályokat kell betartanunk:

- A prefix utasításokat külön sorban kell megadni önálló utasításként.
- A FAR RETURN mnemonikja RETF.
- A stringmozgató utasításokban a mnemonikkal kell megadni az elemek méretét, pl: MVSW, LODSB.
- Az ugrások automatikusan SHORT, NEAR, FAR típusúak lesznek, de ez felülírható a NE[AR] vagy a FAR mnemonikkal.
- A memória hivatkozásoknál a
  WO[RD] PTR és a
  BY[TE] PTR-t
  kell használni, ha az utasításokból ez nem egyértelmű.
- A zárójel nélküli operandusokat immediate (közvetlen) adatnak, a szögletes zárójelbe tetteket memóriahivatkozásnak tekinti a SYMDEB.
- A DB, és DDW direktívák az assemblernek megfelelő módon használhatók.
- Az assemblerben használt minden indexelési írásmód megengedett.
- A mnemonikok szinonímái használhatók, pl: JZ, JE.
- Az FWAIT utasítást külön sorban az előtt az utasítás előtt kell megadni, amelyikre vonatkozik.

Ha szintaktikai hibát vétünk, a SYMDEB az "Error" üzenetet írja ki, és nem fordítja be az utasítást.

10.5.2 A Break Point Set (töréspont megadása) parancs

BP[*n*] *cím* [*m*]

A fenti parancs **állandó töréspontot helyez el** a programban. Egyidőben 10 töréspont lehetséges. Ezek sorszámát jelenti az *n* paraméter. Ha nem adjuk meg, a következő üres töréspont automatikusan hozzárendelődik. **BP** és *n* között **nem lehet szóköz!** A megadott töréspont áthaladás után sem törlődik, mint a GO paranccsal megadott töréspont. Ha az *m* paramétert megadjuk, az első *m* áthaladáskor a töréspont figyelmen kívül marad (*m* egy 16 bites érték). *Az érvényes töréspont elérésekor a SYMDEB kiírja a regiszterek tartalmát,*

a flag-eket és az utasítást, és a prompttal jelentkezik. Ekkor bármilyen SYMDEB parancsot kiadhatunk.

Minden töréspont ki- és bekapcsolható, törölhető és felülírható.

10.5.3 A Breakpoint Clear (töréspont törlése) parancs

BC *lista* | *

Törli a *listá*ban felsorolt sorszámú töréspontokat, illetve az összeset * megadása esetén.

10.5.4 A Breakpoint Disable (töréspont kikapcsolása) parancs

BD *lista* | *

Kikapcsolja a *listá*ban felsorolt sorszámú, illetve az összes töréspontot (* megadása esetén). Ezek a **BE** paranccsal kapcsolhatók vissza.

10.5.5 A Breakpoint Enable (töréspont bekapcsolása) parancs

BE *lista* | *

Bekapcsolja a *listá*ban felsorolt sorszámú, illetve a * megadása esetén az összes töréspontot.

10.5.6 A Breakpoint List (töréspontok kiírása) parancs

BL *lista* | *

Kiírja a *listá*ban felsorolt, vagy az összes töréspontokat a következő formában:

n s cím [m1(m2)]

n a töréspont sorszáma
s a töréspont állapota
e - bekapcsolva
d - kikapcsolva
cím a töréspont címe. Ha van szimbóluma akkor a szimbólummal, ha nincs, számokkal
m1 a töréspont aktivizálódásáig hátralévő áthaladások száma
m2 a töréspont megadásakor specifikált áthaladási szám

m1 és m2 csak akkor szerepel, ha a BP parancsban megadtuk az *m* paramétert.

10.5.7 A Compare (összehasonlítás) parancs

C *címtartomány cím*

összehasonlítja a *címtartomány* terület tartalmát a *cím*en kezdődő terület tartalmával. Ha nem talál eltérést, nem ír ki semmit, illetve ahol eltérés van, kiírja a címeket és a két értéket.

10.5.8 A Display (?) (kifejezés értékének kiírása) parancs

? *kifejezés*

Kiírja a *kifejezés* értékét minden értelmes formában (16 bites, 32 bites hexadecimális, decimális (zárójelben), stringként (idézőjelek közt). A *kifejezés* a 10.4.8 pont szerinti elemekből állhat.

10.5.9 A Dump ASCII (karakteres kiíratás) parancs

DA [*címtartomány*]

A megadott *címtartomány*nak megfelelő memóriatartományt ASCII karakterekként értelmezve kiírja a képernyőre. A vezérlő-karakterek helyett pontot ír. A kiírást az első 0 értékű byte-nál abbahagyja. Ha paramétert nem adunk meg, az előző DA végrehajtásának folytatásaként végzi a kiírást az első 0 értékű byte-ig, de legfeljebb 128 byte-ot ír ki.

10.5.10 A Dump Bytes (byte-os kiíratás) parancs

DB [*címtartomány*]

A megadott *címtartomány* tartalmát hexadecimális és ASCII formátumban kiírja. Az ASCII mezőben a vezérlő-karakterek helyett pontot ír.

10.5.11 A Dump Words (szavas kiíratás) parancs

DW [*címtartomány*]

A megadott címtartomány tartalmát szavakként (az alsó-felső byte-os tárolást átalakítva) kiírja.

10.5.12 A Dump Double words (duplaszavas kiíratás) parancs

DD [*címtartomány*]

A megadott címtartomány tartalmát duplaszóként (szegmens:offset) kiírja.

10.5.13 A Dump Short Reals (valós számos kiíratás) parancs

DS [*címtartomány*]

A megadott *címtartomány* tartalmát rövid valós számokként értelmezve kiírja. Előbb a 4 byte hexadecimális értékét, majd a decimális megfelelőjét írja minden sorba. A decimális számot 16 jegyre írja ki, de ebből természetesen csak az első 7 jegy értékes.

10.5.14 A Dump Long Reals (dupla pontosságú kiíratás) parancs

DL [*címtartomány*]

A megadott *címtartomány* tartalmát hosszú valós számokként értelmezve kiírja. Előbb a 8 byte hexadecimális értékét, majd a decimális megfelelőjét írja ki soronként.

10.5.15 A Dump Ten-Byte Reals (tíz byte-os pontosságú kiíratás) parancs

DT [*címtartomány*]

A megadott *címtartomány* tartalmát 10 byte-os valós számként értelmezve kiírja. Előbb a 10 byte hexadecimális értékét, majd a decimális megfelelőjét írja ki soronként.

10.5.16 A Dump (kiírás folytatása) parancs

D [*címtartomány*]

Az előzőleg használt Dump parancs szerinti formátumban kiírja a *címtartomány*ba eső byte-okat. **A D karakter után legalább egy szóköz megadása kötelező.**

10.5.17 Az Enter (adatbeviteli) parancs

E *cím* [*lista*]

A *cím*mel kezdődő területre **beírja a *listá*ban megadott byte-okat**.

Ha a *listá*t nem adjuk meg, kiírja a <cím> tartalmát és egy pontot, majd vár. Ekkor a következő lehetőségeink vannak:

- Beírunk egy hexadecimális számot, majd az ENTER-t; ekkor felülírtuk a byte-ot.
- Megnyomjuk a szóköz billentyűt: a byte értéke nem változik, és a következő byte-ra lépünk.
- A - jelet írjuk be, ekkor az előző byte-ra lépünk vissza.
- <ENTER>-rel befejezzük az ENTER parancsot.

10.5.18 eXamine Symbol Map (szimbol tábla kiíratása) parancs

X [*]
X ?[*filenév*!][*szegmens*:][*szimbólum*]

A parancs kiírja a betöltött szimbólumokat és címüket. Az **X** parancs paraméter nélkül kiírja az aktuális szegmens nevét. Az **X*** alak kiírja az összes szegmens minden szimbólumának nevét és címét. Az **X?** alak kiírja a megadott szegmensben lévő szimbólum címét. A szegmens és szimbólum megadásakor használhatjuk a '*' karaktert, ez a név hátralévő karaktereit helyettesíti, és kiír minden olyan nevet és címet, ami megfelel az ilyen megadásnak.

10.5.19 A Fill (memória feltöltése) parancs

F *címtartomány lista*

A *címtartománnyal* megadott területet feltölti a *listá*ban megadott tartalommal. A *lista* kevesebb elemet is tartalmazhat, ekkor a teljes listát ismétli a feltöltendő terület végéig.

10.5.20 A Go (program indítása) parancs

G[=*startcím*] [*töréspont*[„„]]

Átadja a vezérlést a *startcím*en kezdődő gépi kódú programnak. A program vagy végigfut, vagy a *töréspont*ok valamelyikén vagy a BP paranccsal megadott töréspontok valamelyikén megáll. Ezután a vezérlés visszakerül a SYMDEB-hez. Maximum tíz *töréspont* adható meg és ha valamelyiket eléri a program, valamennyi töréspont törlődik. Ha a *startcím*et nem adjuk meg, a vezérlés a CS:IP-nek megfelelő címre adódik át.

A Go parancs úgy indítja el a felhasználó programját, hogy a verembe menti a flag-eket és a CS, IP regisztereket majd egy IRET utasítást hajt végre, ezért a veremben legalább 6 byte-nyi helynek kell lenni a Go parancs kiadásakor.

A töréspontokra a SYMDEB egy-egy INT 3 utasítást ír be (a régi kódot előbb kimenti), és ha valamelyik töréspontra ráfut a program, az összes ideiglenes töréspontot törli, és visszaállítja a régi értékeket.

Ha a program végigfut, a SYMDEB a

Program terminated normally

üzenetet írja ki.

10.5.21 A Help (?) (segélykérés) parancs

?

Kiírja a képernyőre a következőket:

```
-?
A [<address>] - assemble
BC[<bp>] - clear breakpoint(s)
BD[<bp>] - disable breakpoint(s)
BE[<bp>] - enable breakpoint(s)
BL[<bp>] - list breakpoint(s)
BP<bp> <address> - set breakpoint
C <range><address> - compare
D[type][<range>] - dump memory
E[type]<address>[<list>] - enter
F <range><list> - fill
G [=<address>[<address>...]] - go
H <value><value> - hexadd
I <value> - input from port
K [<value>] - stack trace
L [<addr>[<drive><rec><rec>]] - load
M <range><address> - move
N <filename>[<filename>...] - name
O <value><byte> - output to port
P [=<address>][<value>] - program step
Q - quit
R [<req>][[=]<value>] - register
S <range><list> - search
S {-P&P+} - source level debugging
```

```
T [=<address>][<value>] - trace
U [<range>] - unassemble
V [<range>] - view source lines
W [<address>[<drive><rec><rec>]] - write
X [?]<symbol> - examine symbols(s)
XO<symbol> - open map/segment
Z <symbol><value>

? <expr> - display expression
! [dos command] - shell escape
. - display current source line
\ - screen flip
> } <device/file> - Redirect output
< { <device/file> - Redirect input
=   <device/file> - Redirect both
* <string> - comment

<expr> ops: + - * / : not seg off by wo dw poi port wport mod
and xor or
<type> : Byte, Word, Doubleword, Asciz, Shortreal, Longreal,
Tenbytereal
```

Ez tartalmazza valamennyi parancs rövid ismertetését.

10.5.22 A Hex (hexadecimális műveletvégzés) parancs

H *kif1 kif2*

Kiírja a *kif1* és *kif2* kifejezések összegét és különbségét hexadecimális formában.

10.5.23 Az Input (byte beolvasása) parancs

I *portcím*

Beolvas egy byte-ot a *portcím*ről és kiírja a képernyőre.

10.5.24 A Load (file betöltése) parancs

L [*cím* [*meghajtó rekord szám*]]

Betölt egy file-t, vagy a lemez megadott logikai szektorától kezdve az előírt számú szektort a megadott *cím*re. Betöltés után BX:CX a betöltött byte-ok számát tartalmazza.

Ha file-t akarunk betölteni, azt előzőleg a Name parancscsal kell megadni. Ha nem adtuk meg, a SYMDEB a DS:005CH címen lévő karaktereket értelmezi filenévnek.

Ha *cím*et megadtunk, a betöltés oda történik, ha nem, a CS:0100H címre. Ha logikai rekordokat akarunk betölteni, meg kell adni a *meghajtó kezdőrekord rekordszám* paramétereket. A SYMDEM az adott meghajtóban levő lemez vagy fix lemez DOS partíciójából olvassa be a szektorokat. A legelső szektor 0. sorszámú.

10.5.25 A Move (memóriacím másolása) parancs

M *címtartomány cím*

A *címtartománnyal* megadott területet a *cím*en kezdődő területre másolja. Ha a területek átfedik egymást, a másolás akkor is korrekt.

10.5.26 A Name (file-név megadása) parancs

N [*filenév*][*argumentum*]

Megadhatjuk a következő Load és Write parancs számára a file nevét. Az *argumentum* karakterek az elindítandó program argumentumai lesznek, mintha a DOS parancssorban adtuk volna meg. Vigyázat! *A SYMDEB a nevet a DS:5CH az argumentumo(ka)t a DS:80H címen helyezi el.*

10.5.27 Az Open Map (XO) (.SYM file megadása) parancs

XO [*filenév*!][*szegmensnév*]

Kijelöli az aktív .SYM file-t és/vagy szegmenst.

10.5.28 Az Output (byte kiírása) parancs

O *portcím byte*

A parancs a megadott *byte*-ot kiírja a *portcím* I/O portra.

10.5.29 A PTrace (gyorsított programvégrehajtás) parancs

P [=*startcím*][*szám*]

Végrehajt *szám* darab gépi kódú utasítást *startcím*től kezdve. Ha a *startcím*et elhagyjuk, CS:IP az alapértelmezés. *szám* elhagyása esetén egy utasítást hajt végre.

Minden utasítás végrehajtása után kiírja a regiszterek és flag-ek értékét, ahogy a T parancs is teszi. Ez az utasítás folyamatosan (nem lépésenként) hajtja végre a hívott szubrutinokat és a software megszakításokat.

(Szegmensregisztert töltő utasításoknál: MOV/POP DS/ES/SS, LDS, LES a következő utasítást is végrehajtja!)

10.5.30 A Quit (vége) parancs

Q

Visszatér a DOS-hoz, befejezve a SYMDEB működését.

10.5.31 Az átirányítási <, >, = parancsok

< *eszköznév*
> *eszköznév*
= *eszköznév*

A DOS valamelyik logikai eszközére kapcsolja át az input és/vagy outputokat, de ez csak a SYMDEB által képernyőre írt, illetve beolvasott adatokra vonatkozik. Olyan programok tesztelésekor hasznos, amelyek megtervezett képernyőre (grafikus képernyőre) dolgoznak.

Az *eszköznév* bármelyik logikai eszköz lehet, általában a COM1 használható erre a célra. Visszakapcsolni a konzolra az =CON paranccsal lehet.

10.5.32 A regiszter parancs

R [*regnév* [*érték*]]

Megnézhetjük és módosíthatjuk a CPU regisztereinek és a flag-eknek az értékét. Ha nem adunk meg regiszter nevet, kiírja a teljes regiszterkészlet állapotát. Ha csak nevet adunk meg, értéket nem, kiírja a régi értéket és várja, hogy megadjuk az újat. Ha értéket is adunk, az a megadott regiszterbe kerül.

A következő regiszternevek használhatók:

AX	BX	CX	DX	SP
BP	SI	DI	DS	ES
SS	CS	IP	PC	F

IP és PC egyaránt az utasítás-számlálót jelenti, F a flag-eket. A flag-ek értékeire a következő elnevezések adhatók meg:

flag	1	0
overflow	OV	NV
direction	DN	UP
ITenable	EI	DI
sign	NG	PL
zero	ZR	NZ
aux carry	AC	NA
parity	PE	PO
carry	CY	NC

10.5.33 A Search (keresés) parancs

S *címtartomány lista*

A megadott *címtartomány*ban megkeresi a *listá*ban megadott minta előfordulásait. A megtalált minták címét kiírja. A mintát megadhatjuk hexadecimális és (idézőjelek közt) karakteres formában is.

10.5.34 A Set Source Mode (forrássorok kiírása) parancs

S [-|+|&]

A Microsoft Pascal, C és FORTRAN használata esetén a forrássorok kiírását kapcsolhatjuk ki és be ezzel a paranccsal.

Az S+ bekapcsolja, az S- kikapcsolja a forrássorok kiírását. Az S& formájú parancs után mind a forrássorokat, mind a visszafordított gépi kódot kiírja.

10.5.35 A Trace (programindítás nyomkövetéssel) parancs

T [=*startcím*] [*szám*]

Működése azonos a P paranccsal (5.29 pont), de ez az utasítás a szubrutinokat és a software megszakításokat is lépésenként hajtja végre.

10.5.36 Az Unassemble (visszafordítás) parancs

U [*címtartomány*]

Kilistázza a *címtartomány*ba eső gépi kódú programot, visszfordítva mnemonikokra az utasításokat. Az S paranccsal beállított üzemmódnak megfelelően a forrássorokat és a visszafordított assembly programot egyaránt ki tudja írni.

10.5.37 A Write (file kiírása) parancs

W [*cím* [*meghajtó rekord szám*]]

Kiír a lemezre *cím*től kezdve BX:CX darab byte-ot az N paranccsal megadott file-ba, illetve a *meghajtó rekord szám* paraméterek megadása esetén a lemez *rekord* logikai szektorától kezdve *szám* darab rekordot ír fel.

10.6. Hibaüzenetek

Ha egy parancsot hibásan adunk meg, a következő sorban egy felfelé mutató nyíllal (|) megjelöli azt a szintatikai egységet, amit már nem tudott értelmezni, és kiírja az "Error" szöveget.

Egyéb hibák esetén a parancs végrehajtása félbeszakad és a következő hibaüzeneteket írja ki a SYMDEB (és újabb parancsra vár):

Bad flag:	Hibás flag. A flag nevét hibásan adtuk meg a parancsban.
Too many breakpoints!	Túl sok töréspont. A Go parancsban több, mint 10 töréspontot adtunk meg.
Bad register!	Az R parancsban hibásan adtuk meg egy regiszter nevét
Double flag!	Az R parancsban egy flag-hez két értéket próbáltunk megadni egyszerre.
Breakpoint list or * expected!	A BC, BD, BL parancsban a töréspontok listáját vagy a *-ot nem adtuk meg.
Error reading .SYM file!	Hiba a SYM file olvasásakor. A megadott .SYM file-t nem tudja beolvasni a SYMDEB. A file üres, vagy hibás.

10.7. A SYMDEB kompatibilis fordítók

A következő assemblerek és fordítók képesek SYMDEB kompatibilis kódot előállítani (melyekből .SYM generálható)

Microsoft

FORTRAN 3.0 vagy nagyobb
PASCAL 3.0 vagy nagyobb
C 2.0 vagy nagyobb
MASM 1.0 vagy nagyobb
BASIC Compiler 1.0 vagy nagyobb
Business BASIC Compiler 1.0 vagy nagyobb

IBM

FORTRAN 2.0 vagy nagyobb
Pascal 2.0 vagy nagyobb
Macro Assembler 1.0 vagy nagyobb
BASIC Compiler 1.0 vagy nagyobb

A forrássorok kiírásához szükséges információkat automatikusan generálják:

Microsoft

FORTRAN 3.0 vagy nagyobb
Pascal 3.0 vagy nagyobb

IBM

FORTRAN 2.0 vagy nagyobb
Pascal 2.0 vagy nagyobb

A Microsoft C fordítónál kapcsolóval kell előírni a forrásszintű hibakereséshez szükséges információk generálását.

11. Kereszthivatkozási-lista készítő program (CREF)

11.1 Bevezetés

A CREF program abc-sorrendben felsorolja a programban használt összes szimbólumot és felsorolja összes előfordulásának helyét. Ennek segítségével gyorsíthatjuk a hibakeresést és könnyen megtalálhatók a deklarált, de nem használt szimbólumok, amelyek zavarják a program érthetőségét.

11.2 A CREF használata

A CREF program az assembler által készített .CRF file-t olvasható szövegfile-lá konvertálja. Ehhez természetesen a MASM indításakor a negyedik paramétert (a .CRF file nevét) meg kell adnunk. Pl:

```
MASM prg.asm,prg.obj,prg.lst,prg.crf
MASM prg,,,prg;
MASM
        Source filename [.ASM]: prg
        Object filename [prg.OBJ]: prg
        Source listing [NUL.LST]:
        Cross reference [NUL.CRF]: prg
```

Ezután a CREF-et elindítva elkészíti az olvasható formátumú keresztreferencia file-t.

A parancs formája:

CREF [*crf-file*[,*ref-file*]]

crf-file a MASM által készített .CRF file neve;
ref-file az olvasható formájú kereszthivatkozási lista file neve.

A file neveknél kiterjesztést nem kötelező megadni, az alapértelmezések .CRF és .REF. A névmegadás szabályai azonosak a MASM-nál megszokottal.

Például:

CREF prg; a PRG.CRF file-ból PRG.REF file-t készít

Ha nem adunk meg file-nevet a CREF is rákérdez:

```
CREF
        Cross reference [.CRF]:prg
        Listing [prg.REF]:
```

Ha a CREF futása közben a <Ctrl-Break>-et lenyomjuk a program befejezi működését és nem hoz létre .REF file-t.

11.3 A kereszthivatkozási-lista

A kereszthivatkozási lista névsorban tartalmazza a forrásszövegben definiált összes szimbólumot és utána felsorolja azokat a sorszámokat, ahol a szimbólumra hivatkoztunk. A definíciós sor számát #-kal jelöli meg. Ha egy szimbólum mellett csak ilyen sorszám szerepel, azt csak definiáltuk, de nem használtuk fel, tehát felesleges! Ha a forrásprogramban TITLE direktívával megadtunk fejlécet, azt itt is kiírja a CREF minden lap tetejére.

11.4 Hibaüzenetek

Ha a CREF futás közben hibát talál, kiírja a következő hibaüzenetet és befejezi működését.

Fatal I/O Error:XX
in: file-név

XX a következő lehet:

101 Rossz meghajtónevet adtunk meg, vagy fatális lemezhiba
105 A meghajtó ajtaja nyitva van vagy a printer nincs bekapcsolva
108 A lemez megtelt
110 A megadott .CRF file-t nem találja.
111 A lemez írásvédett, a .REF file-t nem lehet létrehozni.

103, 104, 106, 112-115 CREF belső hibák.

12. Hibajelzések

12.1 Az assembler hibaüzenetei

A MASM külön jelzi a program (assembler) és az I/O hibákat. A hibák két kategóriáját különbözteti meg: a Warning és a Fatal (figyelmeztető és fatális) hibát.

Figyelmeztető hibának ítéli azokat az eseteket, amikor képes volt az adott utasítást értelmezni, de valami félreértésre okot adó eseményt kíván jelezni. Ekkor az OBJ file-t (ha kértük) létrehozza.

Fatális (végzetes) hibát akkor jelez, ha nem tudja az utasítást értelmezni, vagy ellentmondásba ütközik a fordítás során. Ekkor OBJ file nem keletkezik és esetleg a fordítás is félbeszakad.

A fordítás végén a végzetes és a figyelmeztető hibák számát kiírja a képernyőre és az a lista-file-ba is belekerül.

Az újabb változatok a Fatal Error helyett a Severe Error (komoly hiba) kifejezést használják.

12.1.1 Assembler hibák

Programhiba esetén az assembler kiírja a file nevét, a hibás sor számát, egy hibakódot és a hiba megnevezését.

0: **Blokk egymásba skatulyázási hiba**
Egymásba skatulyázható eljárások, szegmensek, adatstruktúrák, makrók nincsenek korrektül lezárva. (Egy blokkot lezártunk, és ennek egy belső blokkja még nyitva van.)

1: **Felesleges karakterek a sorban**
Az utasítást, direktívát sikerült értelmezni, de a sorban még van értékes karakter. (Általában figyelmeztető hiba.)

2: **Már definiált regiszter**
Az assembler belső logikai hibát talált.

3: **Ismeretlen szimbólum típus**

4: **Szimbólum újradefiniálása**

5: **Többszörösen definiált szimbólum**

6: **Fázishiba**
Valamely cimke értéke a fordítás két menetében nem azonos. Tipikus oka hogy egy SHORT-nak deklarált előrehivatkozó ugrás cimkéje távolabb van 127 byte-nál, vagy egy előrehivatkozó ugrás NEAR típusú és a cimke FAR-nak van deklarálva. Az ilyen hiba megkeresésében a /D kapcsoló (listakészítés mindkét menetben) segíthet.

7: **Már van ELSE ág**
Egy újabb ELSE-t akarunk megnyitni, holott az előzőt még nem zártuk le.

8: **Nincs nyitott feltétel**
ELSE vagy ENDIF előzetes IF nélkül.

9: **Nem definiált szimbólum**

10: **Szintatikai hiba**

11: **Tipusmegadás itt nem megengedett**

12: **Itt csoportnév szükséges**
Az adott helyen csoport (GROUP) név-nek kell állnia, és a fordító más szimbólumot talált.

13: **Az első menetben ismertnek kell lennie**
Az adott hivatkozásnak már az első menetben ismertnek kell lennie. Pl. egy IF utasítás argumentumának kiszámíthatónak kell lennie az első menetben is.

14: **Nem megengedett szimbólum típus**
Egy PUBLIC szimbólumot nem megengedett helyen használ.

15: **A szimbólum már definiálva van**
Egy szimbólumot más típusúnak kíván újradefiniálni.

16: **Foglalt szó**
Pl: MOV nem lehet változónév

17: **Nem megengedett előrehivatkozás**
Az első menet hibajelzése, csak akkor lép fel, ha mindkét menetben listát kértünk.

18: **Itt csak regiszter állhat**
Az adott helyen szimbólum, vagy kifejezés nem megengedett.

19: **Hibás a regiszter típusa**
Ilyen típusú regiszter itt nem használható. Pl: INC CS

20: **Itt szegmens vagy csoport kell**
Az adott helyen csak szegmens vagy csoport (GROUP) adható meg.

21: **A szimbólumnak nincs szegmense**
A SEG utasítást olyan szimbólumra alkalmaztuk, amelynek nincs szegmenscíme.
Pl: CR EQU 13
MOV AX,seg CR

22: **Itt típusmegadás szükséges**
Ezen a helyen csak a WORD, DW, QW, BYTE, TB fordulhat elő.

23: **Már lokálisnak deklarálta**
Egy korábban lokálisnak deklarált szimbólumot EXTERNAL-nak adtunk meg.

24: **A szegmens paraméterei megváltoztak**
Egy ilyen azonosítójú szegmens más paraméterekkel már szerepelt.

25: **Hibás igazítás illetve kombináció megadás**
Paraméterhiba a SEGMENT direktívában.

26: **Többszörösen definiált szimbólumra hivatkozik**

27: **Operandus hiányzik**

28: **Műveleti utasítás (operátor) hiányzik**

29: **Osztás 0-val, vagy túlcsordulás**
A kifejezésben 0-val kellene osztani, vagy a kifejezés értéke túllépi az adott típus értékhatárát.

30: **Léptetés számláló hiba**
A léptetést megadó kifejezés értéke negatív.

31: **Az operandusok eltérő típusúak**
Pl: MOV AX,DL

32: **EXTERNAL itt nem állhat**
Pl: DB M DUP(?) nem megengedett, ha M EXTERNAL

33: **Itt csak rekord mezőnév állhat**

34: **Itt csak rekord vagy mezőnév állhat**

35: **Az operandus méretét meg kell adni**

36: **Itt csak változó, cimke vagy konstans állhat**

37: **Itt csak struktúra mezőnév állhat**

38: **A bal operandus csak szegmens lehet**
A ':' operátor bal operandusa csak szegmens- (csoport-) név vagy szegmensregiszter lehet.

39: **Egy operandus csak konstans lehet**
Az összeadó operátor hibás használatára utal.

40: **Az operandusoknak azonosaknak kell lenniük, vagy egyiküknek konstansnak kell lennie.** (Hibás kivonás kijelölés.)

41: **Egyszerű típus-operandust kell használni.**
(Itt nem megengedett a STRUC, FIELDS, NAMES, BYTE, WORD illetve DW, helyettük változót kell használni.)

42: **Itt csak konstans állhat**
Az adott helyen olyan kifejezésnek kell állnia, amely fordítási időben kiértékelhető.
Pl: MOV AX,2+K
ahol K EXTERNAL szimbólum.

43: **Az operandusnak szegmenssel kell rendelkeznie**
A SEG direktíva hibás használata

44: **Itt adatnak kell szerepelnie**
Kódra hivatkozik olyan helyen, ahol adatnak kell állnia.
Pl: MOV AX,kod-cimke

45: **Itt kód-nak kell szerepelnie**
Adatra hivatkozik olyan helyen, ahol kódnak kell állnia.

46: **Már volt bázisregiszter**
Két bázisregiszter nem lehet egy címben

47: **már volt indexregiszter**
Két indexregiszter nem lehet egy címben.

48: **Itt index- vagy bázisregiszter kell.**
A szögletes zárójelben az utasítás index- vagy bázisregisztert vár.

49: **Itt nem állhat regiszter**
Az utasítás szintaxisa szerint itt nem állhat regiszter

50: **Az érték túl nagy**
Pl: MOV AL,256

51: **Az operandus nincs a kódszegmensben**
Az operandus az aktuális kódszegmensből nem érhető el.

52: **Nem megengedett operandus típus**
Az utasítás ilyen operandussal nem létezik

53: **Túl nagy relatív ugrás**
Feltételes ugrás -128 vagy + 127-nél távolabbi cimkére.

54: **Az index kifejezés nem számítható ki**

55: **Hibás regiszter érték**
A megadott regiszter érték nem írható be az op.kód regiszter mezőjébe, mert nagyobb 7-nél.

56: **Itt közvetlen adat nem használható**
Pl: PUSH 2

57: **Itt ilyen méretű elem nem állhat**
Pl: az SHR operátor nem alkalmazható duplaszóra

58: **Itt byte-regiszter nem használható**
Pl: PUSH AL

59: **A CS regiszter itt nem állhat**
Pl: XCHG CS,AX

60: **Itt csak az AX vagy az AL állhat**

61: **Szegmensregiszter hibás használata**
Pl: MOV DS,00FAH

62: **Hiányzik vagy nem elérhető a kódszegmens**
Elérhetetlen cimkére ugrás.

63: **Hibás operanduskombináció**

64: **NEAR JMP vagy CALL másik szegmensbe**

65: **A cimke szegmense nem módosítható**

66: **A prefix után nem op.kód szerepel**
A REP prefix után nem op.kód következik

67: **Az ES szegmens nem módosítható**
Pl: STOS DS:CEL

68: **Az adat nem érhető el**
Az utasításhoz tartozó szegmensregiszterhez nem az adatnak megfelelő szegmenst rendeltük hozzá.

69: **A kód szegmensen kívül van**
Egy kódot generáló utasítás nincs szegmens belsejében.

70: **BYTE típusú szegmensben EVEN nem lehet**

72: **Az ismétlésszám nem pozitív**
A DUP direktíva előtt 0 vagy negatív szám nem állhat.

73: **Már EXTERNAL-nak deklaráltuk**
Egy korábban EXTERNAL-nak megadott szimbólumot lokálisnak deklaráltunk.

74: **A megadott DUP túl nagy**
A DUP-ok egymásba skatulyázása túl sok a szerkesztőprogramnak.

75: **A ? hibás használata**
Pl: DB ? + 5

76: **Túl sok kezdőérték**
Egy rekordhoz vagy struktúrához túl sok kezdőértéket adtunk meg.

77: **Hibás kezdőérték lista**
A struktúra kezdőértékei nincsenek < > zárójelben.

78: **Nem megengedett direktíva a struktúra-deklarációban**
Struktúradeklarációban csak megjegyzés és definiáló direktíva állhat.

79: **Nem megengedett DUP használat**
Egy struktúra inicializáló utasításban hibásan használta a DUP direktívát.

80: **STRUC inicializáló utasításban mező illegális értékadása**

81: **Hibás típus a struktúra kezdőértékben**

82: **Regiszterre nem lehet előre hivatkozni**

83: **EQV direktíva saját magára hivatkozik**
ALFA EQU ALFA

84: **Ez a 8087 utasítás nincs emulálva**

85: **Hibás file-vég**
vagy az END utasítás hiányzik, vagy valamelyik blokkot nem zárta be.

12.1.2 I/O hibák

Az I/O hibaüzenetek formája:

MASM Error – *hibaüzenet*
in: *filenév*

A hibaüzenetek:

101: **Hard data**	Hibás adat
102: **Device name**	Eszköznév hiba
103: **Operation**	Művelethiba
104: **File system**	File-kezelő hiba
105: **Device offline**	Az eszköz ki van kapcsolva
106: **Lost file**	Sérült file
107: **File name**	File név hiba
108: **Device full**	Az eszköz (lemez) megtelt
109: **Unknown device**	Ismeretlen eszköz
110: **File not found**	A file nincs meg
111: **Protected file**	Írásvédett file
112: **File in use**	A file-t használják
113: **File not open**	A file nincs nyitva
114: **Data format**	Adatformátum hiba
115: **Line too long**	Túl hosszú sor

12.2. A szerkesztő hibaüzenetei

About to generate .EXE file.
Change diskette in drive A: and press ENTER.
Ha a /P kapcsolót megadtuk, mielőtt az .EXE file-t felírná a lemezre, kiírja a fenti üzenetet. Ekkor kell lemezt cserélni és az ENTER gomb lenyomása után az új lemezre írja a szerkesztés eredményét.

Ambiguous switch error: 'X'
Ismeretlen kapcsoló. A parancsban megadott 'X' kapcsolót nem ismeri fel a program. A szerkesztést felfüggeszti.

Array element size mismatch
Tömb elem méret ellentmondás. Egy közös tömb két deklarációja közt ellentmondás van. Assemblyben a 'közös tömb' kategória nem létezik, ezért ez a hiba nem léphet fel.

Attempt to put segment < name > in more than one group in file < filename >
A < name > nevű szegmens két különböző csoporthoz (group) lett hozzárendelve. Korrigálni kell a hibát és újrafordítani a programot.

Bad value for CparMaxAlloc
A /CPARMAXALLOC kapcsolóval megadott érték nincs az [1,65535] intervallumban.

Cannot find library: < filename.LIB >.
Enter new file spec:
A szerkesztő nem találja a < filename > nevű könyvtárat, ezért kéri, hogy a helyette használandót adjuk meg (elérési út és meghajtó is megadható).

Cannot open list file
A directory vagy a lemez megtelt.

Cannot open response file
A megadott válasz-file-t nem találja, hibás file-nevet adtunk meg.

Cannot nest response file
A válasz-file-okat nem lehet egymásbaskatulyázni. (Egy válasz-file-ban másik válasz-file-ra történik hivatkozás).

Cannot open run file
A directory vagy a lemez megtelt

Cannot reopen list file
A felhasználó nem tette vissza a megfelelő diszket, amikor a szerkesztő ezt kérte. A szerkesztést felfüggeszti.

Common area longer than 65536 bytes
A felhasználói program több mint 64K közös változót használ.

Data record too large
A fordító vagy az assembler hibás object modult állított elő. Értesítse a Microsoft céget a garanciajegyen.
(ha van Önnek!)

Dup record too large
A LIDATA record (az object modulban) 512 byte-nál hoszszabb. Egy struktúradefiníció túl komplikált a forrásszövegben. Pl:

ARRAY 10 dup(11 dup(12 dup(13 dup(...)))))
Egyszerűsítsük a struktúrát úgy hogy egy utasítás 512 bytenál kisebb kódot generáljon.

< filename > is not a valid library
A < filename > névvel megadott file nem érvényes (szabályos) könyvtár. A szerkesztést felfüggeszti.

Fixup overflow near < num > in segment < name > in < filename > offset < num >
A hibát a következők idézhetik elő:
1. egy csoport (group) nagyobb 64K-nál.
2. a szerkesztett programban egy CALL SHORT, vagy egy JMP SHORT másik szegmensre hivatkozik (a kódszegmens nagyobb mint 64K)

3. valamelyik cimkéje megegyezik egy könyvtári cimkével
4. egy EXTRN deklaráció valamelyik szegmens belsejében van (tegyük a szegmensen kívülre a deklarációt)

Incorrect DOS version, use DOS 2.0 or later
A LINK csak a 2.0 vagy nagyobb verziószámú DOS-szal képes futni.

Insufficent stack space
Nincs elég memória a program szerkesztéséhez.

Interrupt number exceeds 255
Az /OVERLAYINTERRUPT kapcsolóval megadott szám nagyobb mint 255. Az érvényes tartomány =[4,255].

Invalid numeric switch specification
Hibás szám egy kapcsoló paraméterében.

Invalid object modul
Az object modulok valamelyike hibás, próbálja őket újrafordítani.

NEAR/HUGE conflict
Egy közös változóra eltérő hivatkozások történtek.
Assemblerben ez a hiba nem fordulhat elő.

Nested left parentheses
Az overlay megadása hibás a parancssorban.

No object modules specified
A szerkesztőt object file-név nélkül indítottuk.

Out of space on list file
A lemez megtelt.

Out of space on run file
A lemez, amire a .EXE file kerül, megtelt.

Out of space on scratch file
Az alapértelmezés szerinti meghajtóban lévő lemez megtelt.

Overlay manager symbol already defined/< name >
Valamelyik cimkéje azonos az overlay kezelő cimkéjével. Cserélje ki a cimkét.

Please replace original diskette in drive:A and press ENTER
Ez az üzenet akkor fordul elő, ha a /P kapcsolót használta, a szerkesztő az .EXE file-t felírta a lemezre és kéri vissza az eredeti lemezt. Tegye vissza a lemezt, és nyomja meg az ENTER billentyűt.

Relocation table overflow
Több mint 16384 távoli hivatkozás (FAR JMP vagy CALL) van a programban. Alakítsa át közeli hivatkozásra az ugrásokat ahol csak lehet, és próbálkozzon újból a szerkesztéssel. Pascal és Fortran használata esetén lehet, hogy elég, ha a debug állapotot kikapcsolja.

Segment limit too high
A szerkesztő nem tudja elhelyezni a táblázatait a memóriában, mert túl sok szegmenst írtunk elő. A /SEGMENT kapcsolóval állítsuk kisebbre a limitet (az alapértelmezés 128), vagy szabadítsunk fel helyet a gépben (SIDEKICK, SUPERKEY,...stb.)

Segment size exceeds 64 K
Valamelyik szegmens mérete nagyobb, mint 64 Kbyte.

Symbol table overflow
A szerkesztett program túl sok PUBLIC, EXTRN, SEGMENT, GROUP... stb. globális szimbólumot használ. Csökkentse ezek számát, vonjon össze szegmenseket, csoportokat... stb.

Terminated by user
CTRL-C-vel megszakította a szerkesztő működését.

Too many external symbols in one module
Valamelyik object modulban túl sok külső (EXTRN) hivatkozás van, csökkentse a számukat.

Too many group-, segment- and classnames in one module
Valamelyik object modulban túl sok csoport, szegmens vagy osztálynevet használ. Csökkentse ezek számát.

Too many groups
A szerkesztett program több, mint 9 csoportot (group) tartalmaz, csökkentse számukat.

Too many GRPDEFs in one module
A szerkesztett program valamelyik modulja több, mint 9 csoportot (group) tartalmaz, csökkentse számukat.

Too many segments
A szerkesztett program túl sok szegmenst tartalmaz. A /SEGMENT kapcsolóval állítsa feljebb a limitet.

Too many segments in one module
Valamelyik object modul több, mint 255 szegmenst tartalmaz.
Csökkentse a szegmensek számát.

Too many TYPEDEFs
Valamelyik object modul túl sok típusdefiníciót tartalmaz.

Unexpected end of file on library
A szerkesztő futása közben kicserélte azt a lemezt, amelyik a könyvtárat tartalmazza.

Unexpected end of file on scratch file
A VM.TMP file-t tartalmazó lemezt kicserélte.

Unmatched left parenthesis
Az overlay megadása hibás a parancssorban.

Unmatched right parenthesis
Az overlay megadása hibás a parancssorban.

Unrecognized switch error
Érvénytelen karakter a parancssorban, valamelyik kapcsoló megadása hibás.

VM.TMP in an illegal file name and has been ignored
Object file névként VM.TMP-t adott meg. Ilyen nevű object modult nem szabad használni. Nevezze át.

Warning: no stack segment
A szerkesztett program nem tartalmaz stack kombináció típusú szegmenst. Ez .COM-nak szánt file esetén nem hiba!

Warning: too many local symbols
Sorbarendezett listát kért a szerkesztőtől, de túl sok a lokális szimbólum, ezért a lista nem lesz sorbarendezve.

Warning: too many public symbols
Sorbarendezett listát kért a public szimbólumokról, de túl sok szimbólum létezik, ezért a lista nem lesz sorbarendezett.

EltSECRET

EltSECRET
adatvédelmi rendszer

Termékünk, az EltSECRET, a SW'88 díjazott programjának, az EltGUARD-nak adatvédelemre továbbfejlesztett változata.

IBM PC-n működő adatvédelmi rendszer, mely nemcsak az adatokhoz és programokhoz való hozzáférést szabályozza, hanem adattitkosításra (kódolásra) is lehetőséget ad.

Az EltSECRET rendszer jellemzői:

- 64 hierarchia-szint a fájlok védelmére
- 127 alkönyvtárszint
- 2500 jogosult felhasználó
- 3 különböző bonyolultságú kódolási eljárás
- együttműködés lokális hálózati rendszerekkel
- felhasználói menürendszerek kialakítása
- kiegészítő hardverrel fokozott biztonság

Hardverkiegészítő használata esetén a felhasználók fizikai személyi azonosítót kapnak, jelszavuk mellett ez is kell a bejelentkezéshez. A herdver letiltja a floppyról történő rendszerbetöltést is.

A teljes program ára: **76.000,– Ft** (+ ÁFA).
A DEMO változat ára: **8.000,– Ft** (+ ÁFA).

Egy telephely összes számítógépéhez elegendő egy példány EltSECRET rendszert vásárolni.

Bővebb felvilágosítással, bemutatóval állunk rendelkezésére:

eltsoft

EltSOFT
Számítástechnikai és Reklámszervező GMK
1122 Budapest, Városmajor u. 7–9.
Telefon: 550-962 1536 Budapest, Pf. 390.

I–III. kötet ára: 692,– Ft

SZÁMSZÖV®

SZÁMÍTÁSTECHNIKAI KISSZÖVETKEZET

REXLIB

1116 Budapest XI., Hunyadi J. u. 162
Levélcím: 1430 Budapest, Pf. 16.
Telefon: Titkárság: 667-809
Központ: 665-322

Képernyőkezelő, file-kezelő rezidens rutin- és utility-csomag MS–DOS és NOVELL Netware–286 környezetben fejlesztők számára